ACE Working Group
Internet Draft
Intended status: Standards Track
Expires: December 2015

J. Cuellar
S. Suppan
Siemens AG
Henrich Poehls
Univ. Passau
June 15, 2015

Privacy-Enhanced Tokens for Authorization in ACE
draft-cuellar-ace-pat-priv-enhanced-authz-tokens-00

Status of this Memo

Copyright Notice

Abstract

This specification defines PAT, "Privacy-Enhanced-Authorization-Tokens" or "Pseudonym-based Authorization Tokens", a protocol and a token construction procedure for client authorization in a constrained environment, similar to DCAF [I-D.gerdes-ace-dcaf-authorize].

The tokens can be also used to establish a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes.

Table of Contents

1. Introduction

Three well-known problems in constrained environments are the authorization of clients to access resources on servers, the realization of secure communication between nodes, and the preservation of privacy. The reader is referred for instance to [I-D.gerdes-ace-dcaf-authorize] and [I-D.gerdes-ace-actors], and [KoMa2014].

This draft tackles certain aspects of those three problems. It describes a way of constructing Token Material (Key Material) that can be used by clients and servers (or in some cases, more generally by arbitrary nodes) to create secure channels, provide authentication, in a context similar to ACE-DCAF Tickets. Moreover, the construction can be used to offer user consent (in the sense of privacy) and to create dynamically pseudonyms to enhance the unlinkability of the information, see Subsection "Features" below.

This draft uses the same architecture of [I-D.gerdes-ace-actors], designed to help constrained nodes with authorization-related tasks via less-constrained nodes. As in DCAF, PAT supports an implicit authorization mode where no authorization information is exchanged and uses access tokens to implement this architecture. A device that wants to access an item of interest on a constrained node first has to gain permission in the form of a token from the node’s Authorization Manager.

A main goal of PAT is to securely transmit authorization tokens. A by-product is the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] between two nodes. Notice that the DTLS channel is not needed to securely transmit the authorization tokens. In some cases, relevant in constrained environments, it is also not necessary for a secure transmission of the payload data from server to client.

1.1. Key words to Indicate Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119]. In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

1.2. Features

- The method allows a User, or an Authentication/Authorization Manager on its behalf, to authorize one (or several) client(s) to access resources on a server. The client and/or the server can be constrained devices. The authorization is implemented by distributing purpose-built Key Material (which we generically call "Tokens") to the server and clients. This SHOULD be done by secure channels.

- The Client Tokens are crafted in such a way that the clients can construct authorization tokens that allow them to demonstrate to the server their authorization claims. The message exchange between client and server for the presentation of the tokens MAY be performed via insecure channels.

- Further, the purpose-built Key material and tokens can be used for establishing a secret shared key between a client and the server, which can be then used to establish a DTLS communication with pre-shared keys.

- The tokens do not provide any information about any associated identities or identifiers of the clients nor of the server. In particular, the method can be used in context where unlinkability (privacy) is a main goal: the tokens convey only the assurance of the authorization claims of the clients.

  This means that the payloads of our protocol, and in particular, the Authentication Token secrets used, can be constructed in such a way that they not leak information about the correspondence of messages to the same Client. In other words: if an eavesdropper observes the messages from the different Clients to and from the server, the protocol does not give him information about which messages correspond to the same Client. Of course, other information, like the IP-addresses or the contents themselves of the requests/responses may leak some information in this regard, but that is not information leaked by our protocol and can be treated separately.

- The tokens may be supported by a "proof-of-possession" (PoP) method. PoP allows an authorized entity (a client) to prove to the verifier (here, the server), that he is indeed the intended

authorized owner of the token and not simply the bearer of the token. (Notice that the Authorization Token may be sent in the clear, and thus, it could be stolen by an intruder. A PoP would hinder the attacker to use the token pretending to be authorized).

- o The Key Material can be used to generate and coordinate pseudonyms between C and S and potentially further parties.

- o The user (more precisely, the Resource Owner, RO, see Section " Actors and Terminology" below) is able to decide (if he wishes: in a fine-grained way and in real-time) which client under which circumstances may access his data stored in S. This can be used to provide consent (in terms of privacy) from users (again, ROs).

As DCAF, it has the following features:

- o Simplified authentication on constrained nodes by handing the more sophisticated authentication over to less-constrained devices.

- o Support of secure communication between constrained devices

- o Authorization policies of the principals of both participating parties are ensured.

- o Simplified authorization mechanism for cases where implicit authorization is sufficient.

- o Using only symmetric encryption on constrained nodes.

1.3. Actors and Terminology

The actors and terminology are the same as in DCAF. Very briefly, for the purposes of this draft, the main actors are:

Server (S): An endpoint that hosts and represents a CoAP (see [RFC7252]) resource.

Client (C): An endpoint that attempts to access a CoAP resource on the Server.

Server Authorization Manager (SAM): An entity that prepares and endorses authentication and authorization data for a Server.

Client Authorization Manager (CAM): An entity that prepares and endorses authentication and authorization data for a Client.

Resource Owner (RO): The principal that is in charge of the resource and controls its access permissions. The RO is often the data subject of the protected resource.

In order to avoid confusions, instead of redefining the terms of DCAF, we use additionally the following terms:

Server Token (ST): The token which is generated by the SAM for the Server. Besides parameters, which may contain authorization information that represents RO's authorization policies for C, it contains a secret, St, called the ST-secret. This one can be used to verify the Authorization Token and to generate other secrets to be discussed later.

Client Token (CT): The token which is generated by the SAM for the Client. It contains a secret, Ct, which can be used to generate the Authorization Token, pus some other data used for PoP. Optionally CT may contain authorization information that represents RO's authorization policies for C.

Authorization Token (AT): The token which is generated by the Client and presented by him to the Server. It contains a secret At, which changes regularly (in a similar way to one-time passwords). The AT contains all information needed by the Server to verify that it was granted by SAM.

VerifK, PSK, IntK, ConfK: Derived keys between C and S used respectively:
. to verify that they are talking with the intended partner, for the Client C it is used as Proof of Possession of the (current) Authorization Token
. as Pre-shared Key to establish a DTLS secure channel
. for Integrity protection (in message authentication codes)
. for Confidentiality Protection (to be elaborated in a future version of the document).

2. System Overview

As in DCAF, each Server (S) has a Server Authorization Manger (SAM) which conducts the authentication and authorization for S. S and SAM are assumed to have a secure channel, probably a DTLS channel, but the current specification does not assume anything about it, except that it is two way secure, preserving integrity and confidentiality. Using this secure communication channel SAM provides to S the main secret x which is used within the initial version of the Server Token (ST). Thus ST = (St, paramS), where St is a "main secret" created by SAM (in a way that is outside of the scope of this

draft), and paramS is a set of parameters, determining the functions h, f, r, g1, g2, g3, g4, etc. to be discussed later and, optionally the authorization policies for the clients foreseen.

To gain access to a specific resource on a Server S, a Client (C) requests a token from the SAM, either directly or using its CAM. In the following, for simplicity, we only discuss the collocated CAM-C role; the separation of the roles should be clear to the reader (and will be detailed in subsequent versions of the ID).

After SAM receives the request from C, he decides if C is allowed to access the resource. If so, it generates a Client-Id and a corresponding Client-Token used for the authorization and for securing the communication between C and S.

For explicit access control, SAM adds the detailed access permissions to the token in a way that C (or his CAM) can interpret and S can verify as authentically stemming from SAM.

Then C presents the Authorization Token to S, demonstrating his authorization, and C and S can establish a secure channel.

As in DCAF, an Authorization Manager has to fulfill several requirements regarding enough storage, use interaction and processing power, see [I-D.gerdes-ace-a2a].

3. Protocol Overview

The PAT protocol comprises three parts, see Fig. 1:

1. Transfer of ST = (St, paramS) form SAM to S.

2. Client Token Request from C to SAM and the respective Client Token (CT) grant from SAM to C.

3. Access Requests with their respective Authorization Tokens (AT) between C and S.

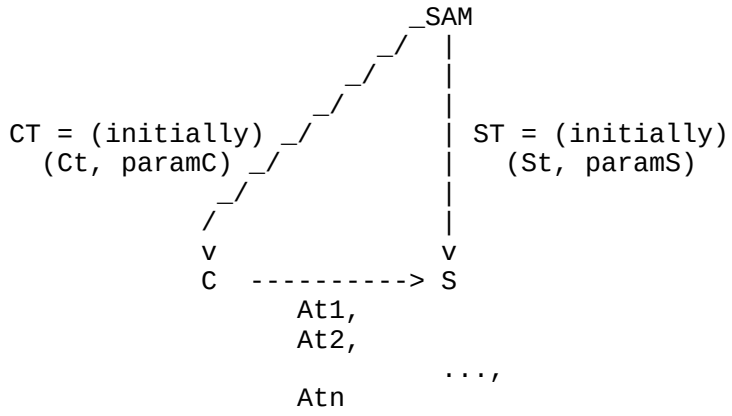


Figure 1: The 3 main parts of the Protocol

There are 3 main Tokens: ST, CT and AT, each of the form (nonce, param), where the nonce is St, Ct, and At, resp., and param = paramC/paramS/paramT is some additional information. (ParamT is not shown in the Figure, for readability).

3.1. Message Flow Overview

In Figure 2, a PAT protocol flow is depicted (messages in square brackets are optional). Notice that in comparison to DCAF, rows 07 and 08 are in different order and the DTLS channel between C and S is optional. The resource response (09) can be optionally secured by DTLS or by other native PAT methods:

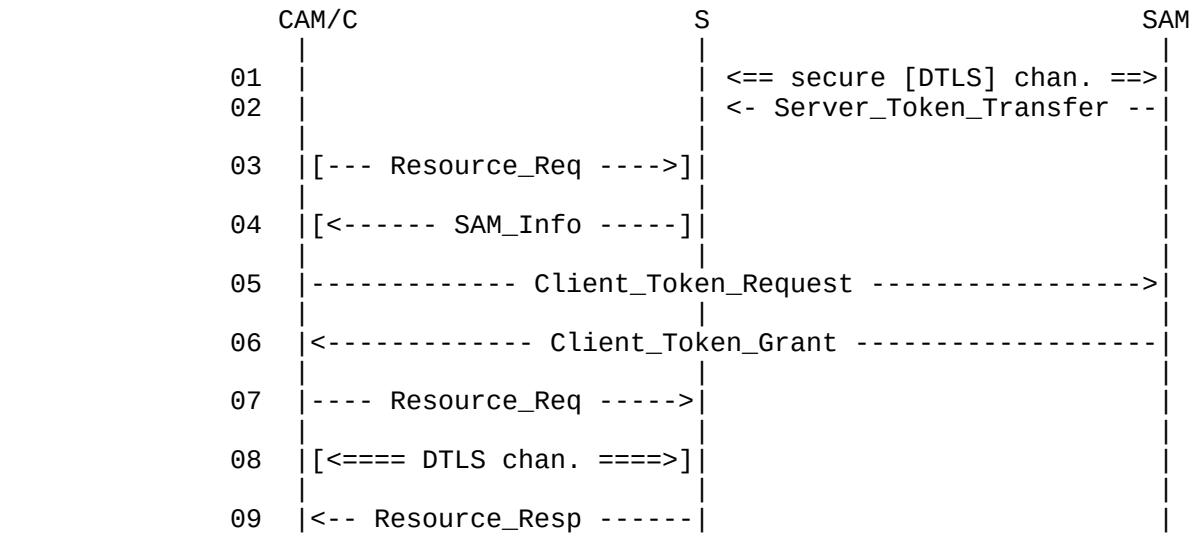


Figure 1: Protocol Overview

As in DCAF, to determine the SAM in charge of a resource hosted at the S, C MAY send an initial Unauthorized Resource Request message to S. S then denies the request and sends the address of its SAM back to C. Or, instead of the initial Unauthorized Resource Request message, C MAY look up the desired resource in a resource directory (cf. [I-D.ietf-core-resource-directory]) that lists the available resources.

Once C knows SAM's address, it can send a request for authorization to SAM (directly, as in Fig. 1 or indirectly using its own CAM). If the access is to be authorized, SAM generates a Client Token (CT) for C. It contains keying material for generating all necessary tokens and keys, and, if necessary, a representation of the permissions C has for the resource.

Each time C sends S a Resource Request, it generates and presents a (current) Authorization Token to S to prove its right to access. With their common knowledge in St and Ct, C and S are able to establish a secure channel.

The following sections specify the message flows in more detail, how the token secrets St, Ct and At are constructed, how the tokens can be revoked, and how S and C can use their common knowledge to verify

the authenticity of the ATs and to obtain a the shared keys VerifK, PSK, IntK, and ConfK.

4. Protocol Details

In the following descriptions the notation

<Msg_Name> A --> B : payload

represents the message with name Msg_Name, sent from A to B and with the given payload.

4.1. Secure [DTLS] channel: S <==> SAM

We assume that the Server S and its Authentication Manager SAM share a secure channel, which may be implemented via USB (and physical security) or via DTLs, etc. We do not assume any particular concrete secure channel, but it must be stressed that the security of the protocol strongly depends on how this security is designed and implemented.

4.2. <Server_Token_Transfer> SAM --> S: ST = (St, paramS)

The owner of the server determines a number N which is (probably) an upper bound on the number of Clients that the Server will simultaneously serve. This number N should not be too high, as the storage and computation effort of the server will increase (linearly) with N. (But the owner may decide any time later to increase or decrease the number N if necessary). Using the secure channel, SAM sends to S the initial value of ST = (St, paramS), where St is a (preferably, random) number that can’t be guessed by an attacker, and paramS is a set of parameters that encode the number N, the choice of functions h, f, r, g1, g2, g3, g4, and the permissions Client Nr "i" has (for each Client i, or for a set of them). The permissions may remain undefined or incomplete and can be extended or modified later anytime. They may also contain validity periods or other restrictions in the Service Level Agreement.

At any later point in time the SAM may change ST, or a part of it: send a new value for St, or change or extend the permissions or change N, the number of expected Clients.

4.3. [<Resource_Req> C --> S: request_params]

The optional Unauthorized Resource Request message is a request for a resource hosted by S for which no proper authorization is granted.

S MUST treat any CoAP request as Unauthorized Resource Request message when any of two following holds:

- S has no valid access token for the sender of the request regarding the requested resource.
- S has a valid access token for the sender of the request, but this does not allow the requested action on the requested resource.

4.4. [<SAM_Info> S --> C: SAM-ID]

As in DCAF, the Server CAN instruct the Client about which SAM to contact.

4.5. <Client_Token_Req> C --> SAM: req_permissions_list

The Client contacts directly or indirectly via the CAM the SAM of his desired Server S, expressing the set of permissions it requests to the resources of the Server S.

4.6. <Client_Token_Grant> SAM --> C: CT = (CT, i [, perm])

SAM decides which Client Number "i" the Client C should have. Each Client will have a different number. The number "i" is an integer between 1 and N, the number of Clients. The choice of value for "i" will depend on which permissions the owner has foreseen and, more importantly, the SAM has encoded as parameters sent to S.

In this message the Client Token CT and the number i are sent.

Optionally, SAM can encode the permissions in this message in a way that the Server S can verify the authenticity of the permission. (Details will be given in a later version of the draft).

4.7. <Resource_Req> C -> S: AT = (At, param),[request_params]

In possession of the Client Token, CT, the Client can construct valid Authorization Tokens, AT, which demonstrates his authorization to access the resources he is requesting.

Regularly, the message Resource_Req has to be sent afresh and a new AT must be used: Client C has to renew his Authorization status at the Server. The frequency in which the Client has to send a new AT can be enforced by C and is determined indirectly the owner of S (or by SAM). This allows a fine-grained control on the service level that the Server will provide to the Client (for instance, on the amount of information of sensor data). We assume that the frequency

of renewal is the same for all Clients, but each Client has a different number of Authorization Tokens it can construct.

Each time a new Resource_Req is sent, a new Authorization Token MAY be needed.

4.8. [DTLS channel: C <==> S]

Optionally, a DTLS channel is constructed using pre-shared key constructed from the common information held by C and S.

4.9. <Resource_Resp> S --> C

The server answers the request of the server, as stipulated by the service description. This message can be

5. Construction of the Tokens

The main construction is an infinite tree with a root, denoted by x. The children of any node are constructed using two functions: a hash function h and a 1-1 function f. The Token secrets St, Ct and At are all values in the tree and thus can be constructed from x using h and f. New versions of any of these secrets, including x, can be constructed in a similar way, but additionally using another function r ("revert") instead of f. Other functions g1, g2, g3, and g4 will be used to generate the derived keys VerifK, PSK, IntK, and ConfK. We assume that h, f, r, g1, g2, g3, and g4 are (or may be) all publicly known functions. That is the security of the protocol SHOULD NOT depend on the secrecy of those functions.

5.1. Main data structure

The main data structure used in this document may be viewed abstractly as a tree of values. Each value is a bit string of a fixed size, which we denote m. But this data structure may be implemented in several different ways, for instance as a set of tables representing the currently relevant parts of the tree.

We use a sequence of integer numbers as indexes for the nodes (values) in the tree. To avoid parentheses, commas, and semicolons we write for instance: "123" for the sequence of 3 numbers "1", "2", and "3". In what follows, in all our examples of integer sequences, we will not use numbers that require 2 or more digits (that is, numbers > 9).

The sequences of integers are used to index values in a tree: x_a is the value at the node with position (address) a. In other words,

the nodes (and their values) are denoted as x_a, where a is a sequence of integer numbers. The tree has a root x (x_a where a = epsilon is the empty sequence). The children of x are x_1, x_2, x_3, ..., x_N, where k = 1..N is a singleton list, that is a list with only one number. If x_a is a node in the tree, then the children of x_a all have the form x_a', where a'=a;i is a the concatenation of a and an integer i. The value x_a'= x_a;i = x_(a;i) is calculated as a hash h of a function f of a and i:

$$x\_a' = x\_a;i = h(f(x\_a, i)).$$

The choice of h and f should not be regarded a secret: they are publicly known parameters of the installation for S. It follows that if an entity knows x_a, the entity is able to calculate all descendants of it, that is, all nodes in the subtree with root x_a. But not vice-versa: since h is a one-way function, the knowledge of x_a;i is not enough to calculate x_a.

Note: "x_a" is read as "x sub a" or "x subindex a". "a" is called the index or address of the node.

Note: Since we also use concatenation of bit strings we need to use parenthesis in that case: x_a;i means x_(a;i), while (x_a);bs means the concatenation of the bit strings (x_a) and bs.

5.1.1. Traversing the Tree

Now we describe a simple procedure for traversing part of the tree, which we now assume of a fixed degree. That is, each node of the tree has either no children or has exactly this amount of children.

Assume that we have a certain "current parent node" x_a and a "current node", a proper descendent of x_a, which may be written as x_a;b. Thus, b is a non empty sequence of integers. For example: the fifth child of the second child of x_a is x_a;b, with b = "25".

Traversing the tree with respect to the current parent node x_a, starting at x_a;b gives the following sequence of nodes (loop):

- First, move up from x_a;b until reaching x_a;b', a direct child of x_a. Thus b' is an integer (or, the same: a sequence of only one integer).
- If x_a;b' is the right-most child of x_a, stop here.
- If x_a;(b'+1) is the right-most child of x_a, then move to its first child (if it exists): x_a;(b'+1);1. If x_a;(b'+1) has no children, stop.
- If x_a;(b'+1) is not right-most child of x_a, then move to it.

Example: assume that the "current parent node" is x_31 and the current node is x_31543. Notice that x_31543 is a descendent of x_31 simply because 31 is a prefix of 31543. Assume that the degree of the tree is 7.

- First, move up: this produces the sequence of values:
  - x_31543, x_3154, x_315
- x_315 is a child of the current parent node and x_316 is not the right-most. Thus we move to it:
  - x_316
- x_316 is a child of the current parent node and the right sibling, x_317, is the right-most. Thus we move to its first child:
  - x_3171
- From there we move to the right until we find the last but one:
  - x_3172, x_3173, x_3174, x_3175, x_3176
- From there we move to the first child of the right sibling and then move right:
  - x_31771, x_31772, x_31773, x_31774, x_31775, x_31776.
- Etc. If the degree of the tree is not one, and each node has children, the sequence is infinite. In any case the sequence can be calculated knowing the current node, the current parent node, h and f. If the degree is one, this corresponds to the one-time-passwords construction scheme.

5.2. Construction of St, Ct and At

The root x of the data structure is the "main secret". It is generated by the SAM as a random or pseudo-random number of m bits (m is a parameter of the system installation and can be chosen by SAM; m is also the length of bit strings that the hash h generates). The method used to construct x is out of our scope, but it should be practically impossible to guess by an attacker, even if he knows in plaintext a sequence of previous or future choices of x.

Initially, St = x

St is sent by the SAM to the Server S in the message Server_Token_Transfer. The value of St at the Server may change if the current value of St is revoked by the SAM. For this, it is not necessary to send a new Server_Token_Transfer. (Details in a future version of the I.D).

The root has N children, one for each foreseen Client. The value x_i, for i=1..N is a secret associated to Client number i, but it is not known by the Client. The values x, x_1, x_2, .., x_N are secrets that never leave the SAM or the Server S and should not be leaked.

The first children of x_1, x_2, .., x_N are the initial values of CT. In other words, for Client number i:

Initially, Ct = x_i1

Ct is sent by the SAM to C(i), the Client number i, in the message Client_Token_Grant. Also in this message, the SAM sends the "current node" (used by C to start a traversal), the depth and the degree of the sub-tree at the node Ct.

The value of Ct at the Server may change if the current value of Ct is revoked by the SAM. If this happens, it is necessary for the Client to obtain a new Client_Token_Grant. (Details in a future version of the I.D).

To create the sequence of Authentication Token secrets, At1, At2, ..., the Client traverses the tree starting at a "current node" (determined by the SAM in the parameters of the Client_Token_Grant message) with the current parent node being the current value of Ct.

6. Formal Syntax

tbd

7. Security Considerations

tbd

8. IANA Considerations

tbd.

9. Conclusions

tbd

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC7252] Shelby, Z., Hartke, K. and Borman, C., "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC6347] Rescorla E. and Modadugu N., "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC4279] Eronen P. and Tschofenig H., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[I-D.gerdes-ace-a2a] Gerdes S., "Managing the Authorization to Authorize in the Lifecycle of a Constrained Device", draft-gerdes-ace-a2a-00 (work in progress), September 2015.

[I-D.gerdes-ace-actors] Gerdes, S., "Actors in the ACE Architecture", draft-gerdes-ace-actors-05 (work in progress), October 2015.

[I-D.gerdes-ace-dcaf-authorize] Gerdes, S., Bergmann, O., and Bormann C., "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02 (work in progress), September 2015.

[I-D.ietf-core-resource-directory] Shelby Z. and Bormann C., "CoRE Resource Directory", draft-ietf-core-resource-directory-02 (work in progress), November 2014.

Informative References

[KoMa2014] Kohnstamm, J. and Madhub, D., "Mauritius Declaration on the Internet of Things", 36th International Conference of Data Protection and Privacy Comissioners, October 2014.

11. Acknowledgments

This draft is the result of collaborative research in the RERUM EU funded project and has been partly funded by the European Commission (Contract No. 609094).

This document was prepared using 2-Word-v2.0.template.dot.

Appendix A.

A.1. Copyright Statement

Authors' Addresses

Jorge Cuellar
Siemens AG
CT RTC ITS

Email: jorge.cuellar @ siemens.com

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: October 31, 2015

S. Gerdes
Universitaet Bremen TZI
L. Seitz
SICS Swedish ICT AB
G. Selander
Ericsson
C. Bormann, Ed.
Universitaet Bremen TZI

April 29, 2015

An architecture for authorization in constrained environments
draft-gerdes-ace-actors-05

Abstract

Constrained-node networks are networks where some nodes have severe constraints on code size, state memory, processing capabilities, user interface, power and communication bandwidth (RFC 7228).

This document provides terminology, and elements of an architecture / a problem statement, for authentication and authorization in these networks.

Table of Contents

1. Introduction

Constrained nodes are small devices with limited abilities which in many cases are made to fulfill a specific simple task. They have limited hardware resources such as processing power, memory, non-volatile storage and transmission capacity and additionally in most cases do not have user interfaces and displays. Due to these constraints, commonly used security protocols are not always easily applicable.

Constrained nodes are expected to be integrated in all aspects of everyday life and thus will be entrusted with vast amounts of data. Without appropriate security mechanisms attackers might gain control over things relevant to our lives. Authentication and authorization mechanisms are therefore prerequisites for a secure Internet of Things.

In some cases authentication and authorization can be addressed by static configuration provisioned during manufacturing or deployment by means of fixed trust anchors and access control lists. This is particularly applicable to siloed, fixed-purpose deployments. However, as the need for flexible access to assets already deployed increases, the legitimate set of authorized entities as well as their privileges cannot be conclusively defined during deployment, without any need for change during the lifetime of the device. Moreover, several use cases illustrate the need for fine-grained access control policies, for which the access control lists concept may not be sufficiently generic.

The limitations of the constrained nodes ask for security mechanisms which take the special characteristics of constrained environments into account; not all constituents may be able to perform all necessary tasks by themselves. In order to meet the security requirements in constrained scenarios, the necessary tasks need to be assigned to logical functional entities.

This document provides some terminology, as well as elements of an architecture to represent the relationships between the logical functional entities involved; on this basis, a problem description for authentication and authorization in constrained-node networks is provided.

1.1. Terminology

Readers are required to be familiar with the terms and concepts defined in [RFC4949], including "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

REST terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. are defined in [RFC7228].

In addition, this document uses the following terminology:

Resource (R): an item of interest which is represented through an interface. It might contain sensor or actuator values or other information.

Constrained node: a constrained device in the sense of [RFC7228].

Actor: A logical functional entity that performs one or more tasks. Multiple actors may be present within a single device or a single piece of software.

Resource Server (RS): An entity which hosts and represents a Resource.

Client (C): An entity which attempts to access a resource on an RS.

Principal: (Used in its English sense here, and specifically as:) An individual that is either RqP or RO or both.

Resource Owner (RO): The principal that is in charge of the resource and controls its access permissions.

Requesting Party (RqP): The principal that is in charge of the Client and controls the requests a Client makes and its acceptance of responses.

Authorization Server (AS): An entity that prepares and endorses authentication and authorization data for a Resource Server.

Client Authorization Server (CAS): An entity that prepares and endorses authentication and authorization data for a Client.

Authenticated Authorization: A synthesis of mechanisms for authentication and authorization.

Note that other authorization architectures such as OAuth [RFC6749] or UMA [I-D.hardjono-oauth-umacore] focus on the authorization problems on the RS side, in particular what accesses to resources the RS is to allow. In this document the term authorization includes this aspect, but is also used for client-side authorization, i.e., more generally to describe allowed interactions with other endpoints.

2. Architecture and High-level Problem Statement

2.1. Elements of an Architecture

This document deals with how to control and protect resource-based interaction between potentially constrained endpoints. The following setting is assumed:

- An endpoint may host functionality of C, RS or both C and RS.
- C in one endpoint requests to access R on a RS in another endpoint.
- A priori, the endpoints do not necessarily have a pre-existing security relationship to each other.
- Either of the endpoints, or both, may be constrained.

Without loss of generality, we focus on the C functionality in one endpoint, which we therefore also call C, accessing the RS functionality in another endpoint, which we therefore also call RS.

More on the security objectives of the constrained level in Section 5.1.

Figure 1: Constrained Level

The authorization decisions at the endpoints are made on behalf of the principals that control the endpoints. To reuse OAuth and UMA terminology, the present document calls C’s controlling principal the Requesting Party (RqP), and calls RS’s controlling principal the

Resource Owner (RO). Each principal makes authorization decisions (possibly encapsulating them into security policies) which the endpoint it controls then enforces.

The specific security objectives will vary, but for any specific version of this scenario will include one or more of:

- Objectives of type 1: No entity not authorized by the RO has access to (or otherwise gains knowledge of) R.
- Objectives of type 2: C is exchanging information with (sending a request to, accepting a response from) a resource only where it can ascertain that RqP has authorized the exchange with R.

Objectives of type 1 require performing authorization on the Resource Server side while objectives of type 2 require performing authorization on the Client side.

More on the security objectives of the principal level in Section 5.2.

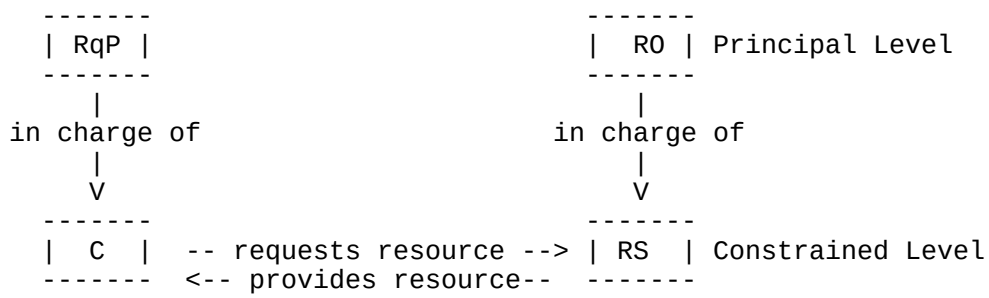


Figure 2: Constrained Level and Principal Level

The use cases defined in [I-D.ietf-ace-usecases] demonstrate that constrained devices are often used for scenarios where their principals are not present at the time of the communication, are not able to communicate directly with the device because of a lack of user interfaces or displays, or may prefer the device to communicate autonomously.

Moreover, constrained endpoints may need support with tasks requiring heavy processing, large memory or storage, or interfacing to humans, such as management of security policies defined by a principal. The principal, in turn, requires some agent maintaining the policies governing how its endpoints will interact.

For these reasons, another level of nodes is introduced in the architecture, the less-constrained level. Using OAuth terminology, AS acts on behalf of the RO to control and support the RS in handling access requests, employing a pre-existing security relationship with RS. We complement this with CAS acting on behalf of RqP to control and support the C in making resource requests and acting on the responses received, employing a pre-existing security relationship with C. To further relieve the constrained level, authorization (and related authentication) mechanisms may be employed between CAS and AS (Section 6.2). (Again, both CAS and AS are conceptual entities controlled by their respective principals. Many of these entities, often acting for different principals, can be combined into a single server implementation; this of course requires proper segregration of the control information provided by each principal.)

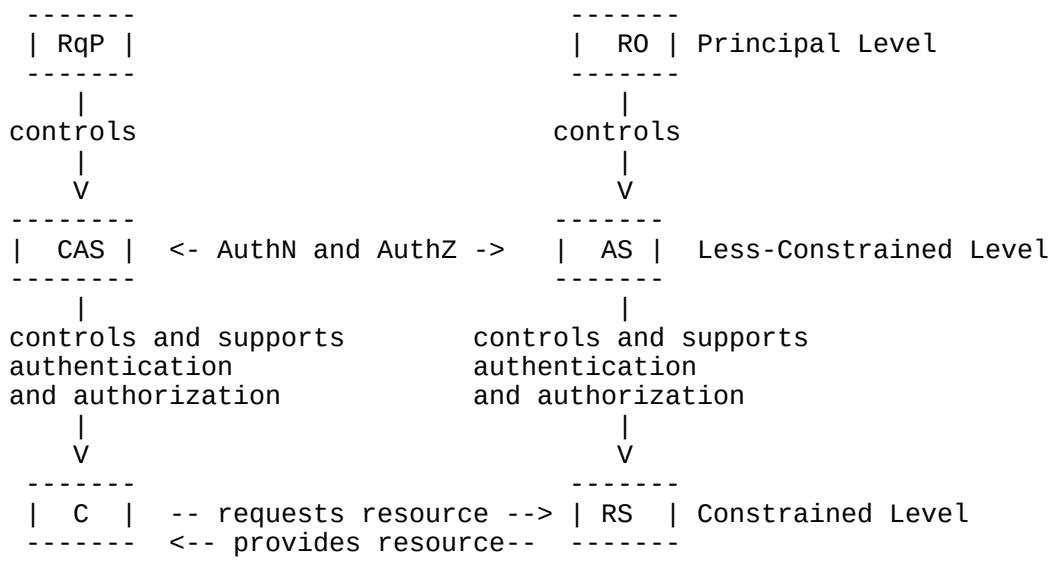


Figure 3: Overall architecture

2.2. Architecture Variants

The elements of the architecture described above are architectural. In a specific scenario, several elements can share a single device or even be combined in a single piece of software. If C is located on a more powerful device, it can be combined with CAS:

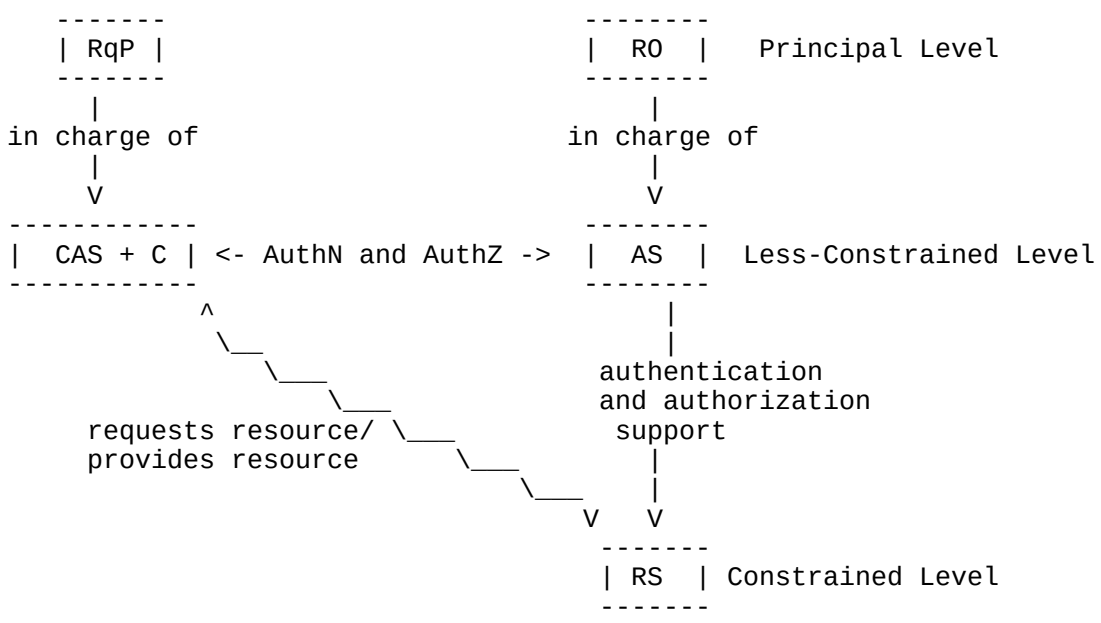


Figure 4: Combined C and CAS

If RS is located on a more powerful device, it can be combined with AS:

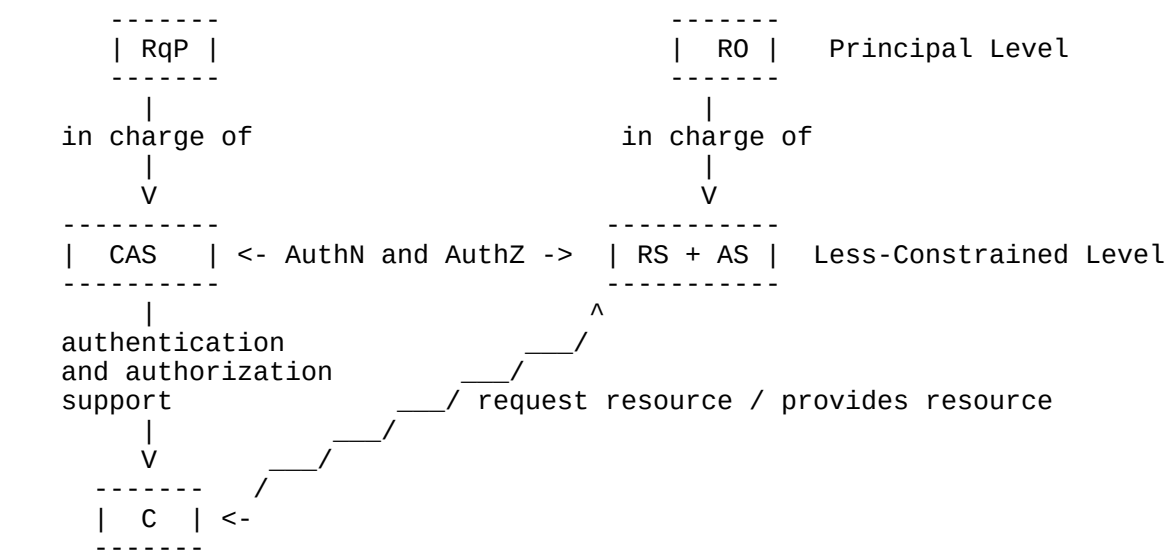


Figure 5: Combined AS and RS

If C and RS have the same principal, CAS and AS can be combined.

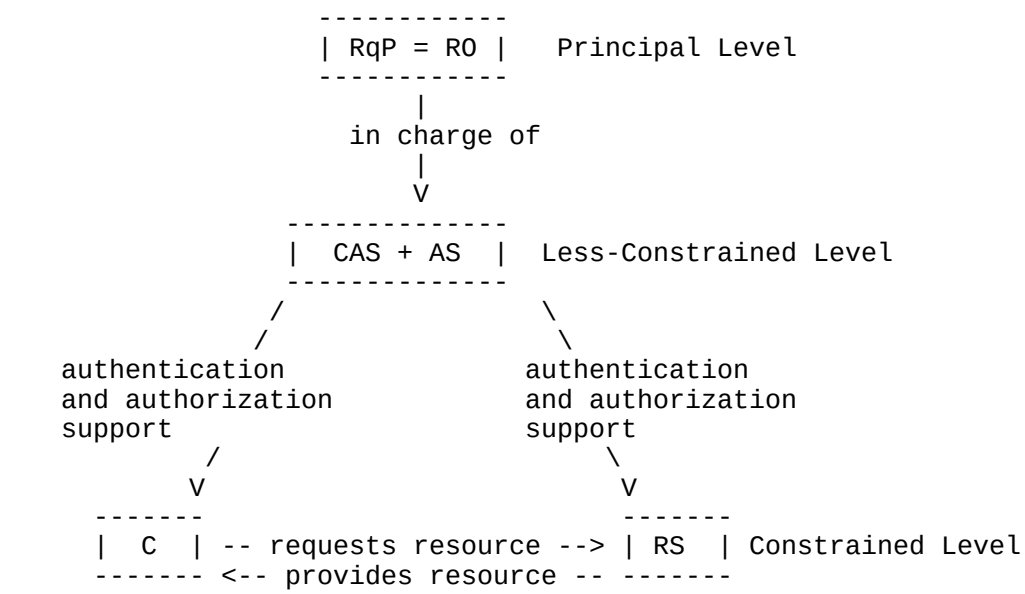


Figure 6: CAS combined with AS

2.3. Information flows

In this subsection, we complement the abstracted architecture described above with a discussion of the information flows in scope, mentioning that each endpoint may assume both a client and a server role and that communication may be via intermediaries.

The less-constrained nodes, CAS and AS, control the interactions between the endpoints by supporting the potentially constrained nodes with control information, for example permissions of clients, conditions on resources, attributes of client and resource servers, keys and credentials. The control information may be rather different for C and RS, reflecting the intrinsic asymmetry with C initiating the request for access to a resource, and RS acting on a received request, and C finally acting on the received response.

The information flows are shown in Figure 7. The arrows with control information only indicate origin and destination of information, actual message flow may pass intermediary nodes (both nodes that are identified in the architecture and other nodes).

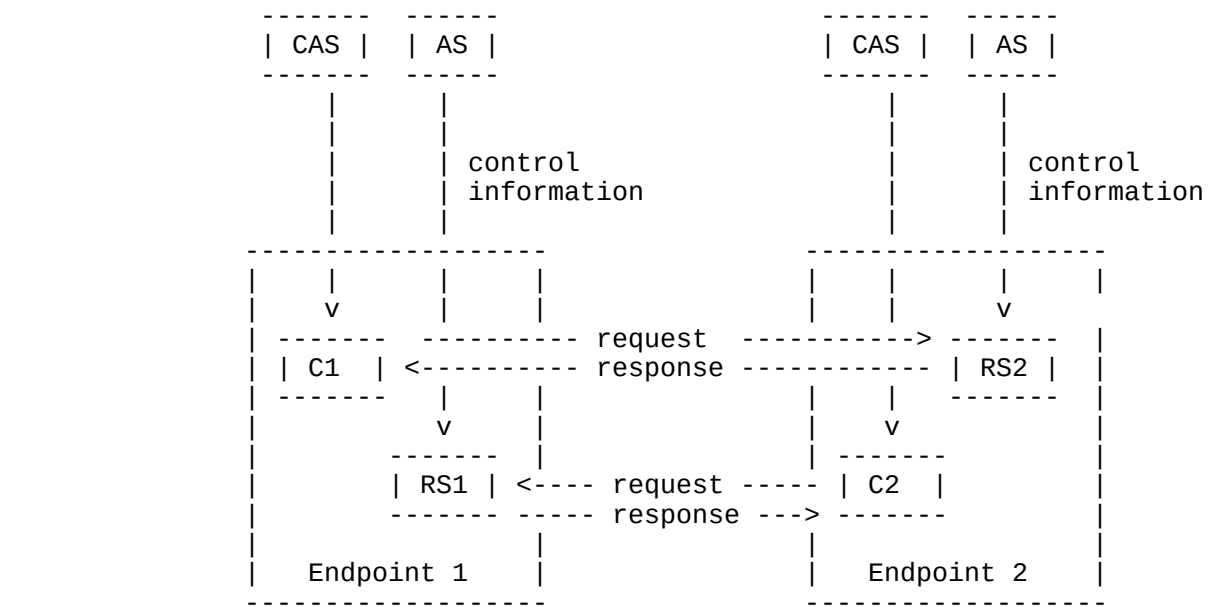


Figure 7: Information flows that need to be protected

- We assume that the necessary keys/credentials for protecting the control information between the potentially constrained nodes and

their associated less-constrained nodes are pre-established, for example as part of the commissioning procedure.

- The messages between the endpoints also need to be protected, potentially end-to-end through intermediary nodes (Section 3.1). Any necessary keys/credentials for protecting the interaction between the endpoints will need to be established and maintained as part of a solution.

2.4. Problem statement

The problem statement for authorization in constrained environments can be summarized as follows:

- The interaction between potentially constrained endpoints is controlled by control information provided by less-constrained nodes on behalf of the principals of the endpoints.

- The interaction between the endpoints needs to be secured, as well as the establishment of the necessary keys for securing the interaction, potentially end-to-end through intermediary nodes.

- The mechanism for transferring control information needs to be secured, potentially end-to-end through intermediary nodes. Pre-established keying material may need to be employed for establishing the keys used to protect these information flows.

3. Security Objectives

The security objectives that are addressed by an authorization solution include confidentiality and integrity. Additionally, allowing only selected entities limits the burden on system resources, thus helping to achieve availability. Misconfigured or wrongly designed authorization solutions can result in availability breaches: Users might no longer be able to use data and services as they are supposed to.

Authentication mechanisms can achieve additional security objectives such as non-repudiation and accountability. These additional objectives are not related to authorization and thus are not in scope of this draft, but may nevertheless be relevant. Non-repudiation and accountability may require authentication on a device level, if it is necessary to determine which device performed an action. In other cases it may be more important to find out who is responsible for the device’s actions.

The security objectives and their relative importance differ for the various constrained environment applications and use cases [I-D.ietf-ace-usecases].

In many cases, one participating party has different security objectives than another. To achieve a security objective of one party, another party may be required to provide a service. E.g., if RqP requires the integrity of representations of a resource R that RS is hosting, both C and RS need to partake in integrity-protecting the transmitted data. Moreover, RS needs to protect any write access to this resource as well as to relevant other resources (such as configuration information, firmware update resources) to prevent unauthorized users from manipulating R.

3.1. End-to-End Security Objectives

In many cases, the information flows described in Section 2.3 need to be protected end-to-end. For example, AS may not be connected to RS (or may not want to exercise such a connection), relying on C for transferring authorization information. As the authorization information is related to the permissions granted to C, C must not be in a position to manipulate this information, which therefore requires integrity protection on the way between AS and RS.

As another example, resource representations sent between endpoints may be stored in intermediary nodes, such as caching proxies or pub-sub brokers. Where these intermediaries cannot be relied on to fulfill the security objectives of the endpoints, these will need to protect the exchanges end-to-end.

Note that there may also be cases of intermediary nodes that very much partake in the security objectives to be achieved. What is the endpoint to which communication needs end-to-end protection is defined by the use case.

In order to support the required communication and application security, keying material needs to be established between the relevant nodes in the architecture.

4. Authentication and Authorization

Server-side authorization solutions aim at protecting the access to items of interest, e.g. hardware or software resources or data: They enable the resource owner to control who can access it and how.

To determine if an entity is authorized to access a resource, an authentication mechanism is needed. According to the Internet Security Glossary [RFC4949], authentication is "the process of

verifying a claim that a system entity or system resource has a certain attribute value." Examples for attribute values are the ID of a device, the type of the device or the name of its owner.

The security objectives the authorization mechanism aims at can only be achieved if the authentication and the authorization mechanism work together correctly. We speak of authenticated authorization to refer to the required synthesis of mechanism for authentication and authorization.

Where used for authorization, the set of authenticated attributes must be meaningful for this purpose, i.e., authorization decisions must be possible based on these attributes. If the authorization policy assigns permissions to an individual entity, the set of authenticated attributes must be suitable to uniquely identify this entity.

In scenarios where devices are communicating autonomously there is often less need to uniquely identify an individual device: For a principal, the fact that a device belongs to a certain company or that it has a specific type (e.g. light bulb) or location may be more important than that it has a unique identifier.

(As a special case for the authorization of read access to a resource, RS may simply make an encrypted representation available to anyone [OSCAR]. In this case, controlling read access to that resource can be reduced to controlling read access to the key; partially removing access also requires a timely update of the key for RS and all participants still authorized.)

Principals (RqP and RO) need to decide about the required level of granularity for the authorization. For example, we distinguish device authorization from owner authorization, and flat authorization from unrestricted authorization. In the first case different access permissions are granted to individual devices while in the second case individual owners are authorized. If flat authorization is used, all authenticated entities are implicitly authorized and have the same access permissions. Unrestricted authorization for an item of interest means that no authorization mechanism is used for accessing this resource (not even by authentication) and all entities are able to access the item as they see fit (note that an authorization mechanism may still be used to arrive at the decision to employ unrestricted authorization).

More fine-grained authorization does not necessarily provide more security but can be more flexible. Principals need to consider that an entity should only be granted the permissions it really needs

(principle of least privilege), to ensure the confidentiality and integrity of resources.

For all cases where an authorization solution is needed (all but Unrestricted Authorization), the enforcing party needs to be able to authenticate the party that is to be authorized. Authentication is therefore required for messages that contain (or otherwise update) representations of an accessed item. More precisely: The enforcing party needs to make sure that the receiver of a message containing a representation is authorized to receive it, both in the case of a client sending a representation to a server and vice versa. In addition, it needs to ensure that the actual sender of a message containing a representation is indeed the one authorized to send this message, again for both the client-to-server and server-to-client case. To achieve this, integrity protection of these messages is required: Authenticity cannot be assured if it is possible for an attacker to modify the message during transmission.

In some cases, only one side (client or server side) requires the integrity and / or confidentiality of a resource value. Principals may decide to omit authentication (unrestricted authorization), or use flat authorization (just employing an authentication mechanism). However, as indicated in Section 3, the security objectives of both sides must be considered, which can often only be achieved when the the other side can be relied on to perform some security service.

5. Actors and their Tasks

This and the following section look at the resulting architecture from two different perspectives: This section provides a more detailed description of the various "actors" in the architecture, the logical functional entities performing the tasks required. The following section then will focus on the protocols run between these functional entities.

For the purposes of this document, an actor consists of a set of tasks and additionally has a security domain (client domain or server domain) and a level (constrained, principal, less-constrained). Tasks are assigned to actors according to their security domain and required level.

Note that actors are a concept to understand the security requirements for constrained devices. The architecture of an actual solution might differ as long as the security requirements that derive from the relationship between the identified actors are considered. Several actors might share a single device or even be combined in a single piece of software. Interfaces between actors

may be realized as protocols or be internal to such a piece of software.

5.1. Constrained Level Actors

As described in the problem statement (see Section 2), either C or RS or both of them may be located on a constrained node. We therefore define that C and RS must be able to perform their tasks even if they are located on a constrained node. Thus, C and RS are considered to be Constrained Level Actors.

C performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including access requests.
- Validate that an entity is an authorized server for R.

RS performs the following tasks:

- Communicate in a secure way (provide for confidentiality and integrity of messages), including responses to access requests.
- Validate the authorization of the requester to access the requested resource as requested.

R is an item of interest such as a sensor or actuator value. R is considered to be part of RS and not a separate actor. The device on which RS is located might contain several resources of different ROs. For simplicity of exposition, these resources are described as if they had separate RS.

As C and RS do not necessarily know each other they might belong to different security domains.

(See Figure 8.)

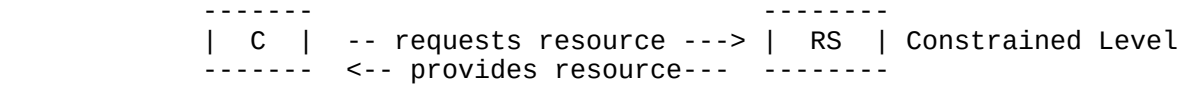


Figure 8: Constrained Level Actors

5.2. Principal Level Actors

Our objective is that C and RS are under control of principals in the physical world, the Requesting Party (RqP) and the Resource Owner (RO) respectively. The principals decide about the security policies of their respective endpoints and belong to the same security domain.

RqP is in charge of C, i.e. RqP specifies security policies for C, e.g. with whom C is allowed to communicate. By definition, C and RqP belong to the same security domain.

RqP must fulfill the following task:

- Configure for C authorization information for sources for R.

RO is in charge of R and RS. RO specifies authorization policies for R and decides with whom RS is allowed to communicate. By definition, R, RS and RO belong to the same security domain.

RO must fulfill the following task:

- Configure for RS authorization information for accessing R.

(See Figure 2.)

5.3. Less-Constrained Level Actors

Constrained level actors can only fulfill a limited number of tasks and may not have network connectivity all the time. To relieve them from having to manage keys for numerous endpoints and conducting computationally intensive tasks, another complexity level for actors is introduced. An actor on the less-constrained level belongs to the same security domain as its respective constrained level actor. They also have the same principal.

The Client Authorization Server (CAS) belongs to the same security domain as C and RqP. CAS acts on behalf of RqP. It assists C in authenticating RS and determining if RS is an authorized server for R. CAS can do that because for C, CAS is the authority for claims about RS.

CAS performs the following tasks:

- Validate on the client side that an entity has certain attributes.
- Obtain authorization information about an entity from C’s principal (RqP) and provide it to C.

- Negotiate means for secure communication to communicate with C.

The Authorization Server (AS) belongs to the same security domain as R, RS and RO. AS acts on behalf of RO. It supports RS by authenticating C and determining C's permissions on R. AS can do that because for RS, AS is the authority for claims about C.

AS performs the following tasks:

- Validate on the server side that an entity has certain attributes.
- Obtain authorization information about an entity from RS' principal (RO) and provide it to RS.
- Negotiate means for secure communication to communicate with RS.

6. Kinds of Protocols

Devices on the less-constrained level potentially are more powerful than constrained level devices in terms of processing power, memory, non-volatile storage. This results in different characteristics for the protocols used on these levels.

6.1. Constrained Level Protocols

A protocol is considered to be on the constrained level if it is used between the actors C and RS which are considered to be constrained (see Section 5.1). C and RS might not belong to the same security domain. Therefore, constrained level protocols need to work between different security domains.

Commonly used Internet protocols can not in every case be applied to constrained environments. In some cases, tweaking and profiling is required. In other cases it is beneficial to define new protocols which were designed with the special characteristics of constrained environments in mind.

On the constrained level, protocols need to address the specific requirements of constrained environments. Examples for protocols that consider these requirements is the transfer protocol CoAP (Constrained Application Protocol) [RFC7252] and the Datagram Transport Layer Security Protocol (DTLS) [RFC6347] which can be used for channel security.

Constrained devices have only limited storage space and thus cannot store large numbers of keys. This is especially important because constrained networks are expected to consist of thousands of nodes.

Protocols on the constrained level should keep this limitation in mind.

6.1.1. Cross Level Support Protocols

Protocols which operate between a constrained device on one side and the corresponding less-constrained device on the other are considered to be (cross level) support protocols. Protocols used between C and CAS or RS and AS are therefore support protocols.

Support protocols must consider the limitations of their constrained endpoint and therefore belong to the constrained level protocols.

6.2. Less-Constrained Level Protocols

A protocol is considered to be on the less-constrained level if it is used between the actors CAS and AS. CAS and AS might belong to different security domains.

On the less-constrained level, HTTP [RFC7230] and Transport Layer Security (TLS) [RFC5246] can be used alongside or instead of CoAP and DTLS. Moreover, existing security solutions for authentication and authorization such as the OAuth web authorization framework [RFC6749] and Kerberos [RFC4120] can likely be used without modifications and there are no limitations for the use of a Public Key Infrastructure (PKI).

7. Elements of a Solution

Without anticipating specific solutions, the following considerations may be helpful in discussing them.

7.1. Authorization

The core problem we are trying to solve is authorization. The following problems related to authorization need to be addressed:

- AS needs to transfer authorization information to RS and CAS needs to transfer authorization information to C.
- The transferred authorization information needs to follow a defined format and encoding, which must be efficient for constrained devices, considering size of authorization information and parser complexity.
- C and RS need to be able to verify the authenticity of the authorization information they receive. Here as well, there is a trade-off between processing complexity and deployment complexity.

- The RS needs to enforce the authorization decisions of the AS, while C needs to abide with the authorization decisions of the CAS. The authorization information might require additional policy evaluation (e.g. matching against local access control lists, evaluating local conditions). The required "policy evaluation" at the constrained actors needs to be adapted to the capabilities of the devices implementing them.

- Finally, as is indicated in the previous bullet, for a particular authorization decision there may be different kinds of authorization information needed, and these pieces of information may be transferred to C and RS at different times and in different ways prior to or during the client request.

7.2. Authentication

The following problems need to be addressed, when considering authentication:

- RS needs to authenticate AS, and C needs to authenticate CAS, to ensure that the authorization information and related data comes from the correct source.

- CAS and AS may need to to authenticate each other, both to perform the required business logic and to ensure that CAS gets security information related to the resources from the right source.

- In some use cases RS needs to authenticate some property of C, in order to map it to the relevant authorization information. In other use cases, authentication and authorization of C may be implicit, e.g. by encrypting the resource representation the RS only providing access to those who possess the key to decrypt.

- C may need to authenticate RS, in order to ensure that it is interacting with the right resources. Alternatively C may just verify the integrity of a received resource representation.

- CAS and AS need to authenticate their communication partner (C or RS), in order to ensure it serves the correct device.

7.3. Communication Security

There are different alternatives to provide communication security, and the problem here is to choose the optimal one for each scenario. We list the available alternatives:

- Session-based security at transport layer such as DTLS [RFC6347] offers security, including integrity and confidentiality

protection, for the whole application layer exchange. However, DTLS may not provide end-to-end security over multiple hops. Another problem with DTLS is the cost of the handshake protocol, which may be too expensive for constrained devices especially in terms of memory and power consumption for message transmissions.

- An alternative is object security at application layer, e.g. using [I-D.selander-ace-object-security] Secure objects can be stored or cached in network nodes and provide security for a more flexible communication model such as publish/subscribe (compare e.g. CoRE Mirror Server [I-D.koster-core-coap-pubsub]). A problem with object security is that it can not provide confidentiality for the message headers.

- Hybrid solutions using both session-based and object security are also possible. An example of a hybrid is where authorization information and cryptographic keys are provided by AS in the format of secure data objects, but where the resource access is protected by session-based security.

7.4. Cryptographic Keys

With respect to cryptographic keys, we see the following problems that need to be addressed:

Symmetric vs Asymmetric Keys
: We need keys both for protection of resource access and for protection of transport of authentication and authorization information. Do we want to support solutions based on asymmetric keys or symmetric keys in both cases? There are classes of devices that can easily perform symmetric cryptography, but consume considerably more time/battery for asymmetric operations. On the other hand asymmetric cryptography has benefits e.g. in terms of deployment.

Key Establishment
: How are the corresponding cryptographic keys established? Considering Section 7.1 there must be a mapping between these keys and the authorization information, at least in the sense that AS must be able to specify a unique client identifier which RS can verify (using an associated key). One of the use cases of [I-D.ietf-ace-usecases] describes spontaneous change of access policies - e.g. giving a hitherto unknown client the right to temporarily unlock your house door. In this case C is not previously known to RS and a key must be provisioned by AS.

Revocation and Expiration

How are keys replaced and how is a key that has been compromised revoked in a manner that reaches all affected parties, also keeping in mind scenarios with intermittent connectivity?

8. Assumptions and Requirements

In this section we list a set of candidate assumptions and requirements to make the problem description in the previous sections more concise and precise.

8.1. Architecture

The architecture consists of at least the following types of nodes:

- RS hosting resources, and responding to access requests
- C requesting access to resources
- AS supporting the access request/response procedure by providing authorization information to RS
  - AS may support this by aiding RS in authenticating C, or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- CAS supporting the access request/response procedure by providing authorization information to C
  - CAS may support this by aiding C in authenticating RS, forwarding information between AS and C (possibly ultimately for RS), or providing cryptographic keys or credentials to C and/or RS to secure the request/response procedure.
- The architecture allows for intermediary nodes between any pair of C, RS, AS, and CAS, such as forward or reverse proxies in the CoRE architecture. (Solutions may or may not support all combinations.)
  - The architecture does not make a choice between session based security and data object security.

8.2. Constrained Devices

- C and/or RS may be constrained in terms of power, processing, communication bandwidth, memory and storage space, and moreover:
  - unable to manage complex authorization policies

    * unable to manage a large number of secure connections

    * without user interface

    * without constant network connectivity

    * unable to precisely measure time

    * required to save on wireless communication due to high power consumption

  o CAS and AS are not assumed to be constrained devices.

  o All devices under consideration can process symmetric cryptography without incurring an excessive performance penalty.

    * We assume the use of a standardized symmetric key algorithm, such as AES.

    * Except for the most constrained devices we assume the use of a standardized cryptographic hash function such as SHA-256.

  o Public key cryptography requires additional resources (e.g. RAM, ROM, power, specialized hardware).

  o A DTLS handshake involves significant computation, communication, and memory overheads in the context of constrained devices.

    * The RAM requirements of DTLS handshakes with public key cryptography are prohibitive for certain constrained devices.

    * Certificate-based DTLS handshakes require significant volumes of communication, RAM (message buffers) and computation.

  o A solution will need to consider support for a simple scheme for expiring authentication and authorization information on devices which are unable to measure time (cf. section Section 9.2).

8.3. Authentication

  o RS needs to authenticate AS to ensure that the authorization information and related data comes from the correct source.

  o Similary, C needs to authenticate CAS to ensure that the authorization information and related data comes from the correct source.

- Depending on use case and authorization requirements, C, RS, CAS, or AS may need to authenticate messages from each other.

8.4. Server-side Authorization

- RS enforces authorization for access to a resource based on credentials presented by C, the requested resource, the REST method, and local context in RS at the time of the request, or on any subset of this information.
- The credentials presented by C may have been provided by CAS.
- The underlying authorization decision is taken either by AS or RS.
- The authorization decision is enforced by RS.
  - RS needs to have authorization information in order to verify that C is allowed to access the resource as requested.
  - RS needs to make sure that it provides resource access only to authorized clients.
- Apart from authorization for access to a resource, authorization may also be required for access to information about a resource (e.g. resource descriptions).
- The solution may need to be able to support the delegation of access rights.

8.5. Client-side Authorization Information

- C enforces client-side authorization by protecting its requests to RS and by authenticating results from RS, making use of decisions and policies as well as keying material provided by CAS.

8.6. Server-side Authorization Information

- Authorization information is transferred from AS to RS using Agent, Push or Pull mechanisms [RFC2904].
- RS needs to authenticate that the authorization information is coming from AS (integrity).
- The authorization information may also be encrypted end-to-end between AS and RS (confidentiality).
- The architecture supports the case where RS may not be able to communicate with AS at the time of the request from C.

- RS may store or cache authorization information.
- Authorization information may be pre-configured in RS.
- Authorization information stored or cached in RS needs to be possible to change. The change of such information needs to be subject to authorization.
- Authorization policies stored on RS may be handled as a resource, i.e. information located at a particular URI, accessed with RESTful methods, and the access being subject to the same authorization mechanics. AS may have special privileges when requesting access to the authorization policy resources on RS.
- There may be mechanisms for C to look up the AS which provides authorization information about a particular resource.

8.7. Resource Access

- Resources are accessed in a RESTful manner using GET, PUT, POST, DELETE.
- By default, the resource request needs to be integrity protected and may be encrypted end-to-end from C to RS. It needs to be possible for RS to detect a replayed request.
- By default, the response to a request needs to be integrity protected and encrypted end-to-end from RS to C. It needs to be possible for C to detect a replayed response.
- RS needs to be able to verify that the request comes from an authorized client
- C needs to be able to verify that the response to a request comes from the intended RS.
- There may be resources whose access need not be protected (e.g. for discovery of the responsible AS).

8.8. Keys and Cipher Suites

- A constrained node and its authorization manager (i.e., RS and AS, and C and CAS) have established cryptographic keys. For example, they share a secret key or each have the other’s public key.
- The transfer of authorization information is protected with symmetric and/or asymmetric keys.

- The access request/response can be protected with symmetric and/or asymmetric keys.
- There must be a mechanism for RS to establish the necessary key(s) to verify and decrypt the request and to protect the response.
- There must be a mechanism for C to establish the necessary key(s) to protect the request and to verify and decrypt the response.
- There must be a mechanism for C to obtain the supported cipher suites of a RS.

8.9. Network Considerations

- A solution will need to consider network overload due to avoidable communication of a constrained node with its authorization manager (C with CAS, RS with AS).
- A solution will need to consider network overload by compact authorization information representation.
- A solution may want to optimize the case where authorization information does not change often.
- A solution may consider support for an efficient mechanism for providing authorization information to multiple RSs, for example when multiple entities need to be configured or change state.

8.10. Legacy Considerations

- A solution may consider interworking with existing infrastructure.
- A solution may consider supporting authorization of access to legacy devices.

9. Security Considerations

This document discusses authorization-related tasks for constrained environments and describes how these tasks can be mapped to actors in the architecture.

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments are provided in e.g. OAuth 2.0 [RFC6749].

In this section we focus on specific security aspects related to authorization in constrained-node networks. Section 11.6 of [RFC7252], "Constrained node considerations", discusses implications

of specific constraints on the security mechanisms employed. A wider view of security in constrained-node networks is provided in [I-D.garcia-core-security].

9.1. Physical Attacks on Sensor and Actuator Networks

The focus of this work is on constrained-node networks consisting of connected sensors and actuators. The main function of such devices is to interact with the physical world by gathering information or performing an action. We now discuss attacks performed with physical access to such devices.

The main threats to sensors and actuator networks are:

- Unauthorized access to data to and from sensors and actuators, including eavesdropping and manipulation of data.

- Denial-of-service making the sensor/actuator unable to perform its intended task correctly.

  A number of attacks can be made with physical access to a device including probing attacks, timing attacks, power attacks, etc. However, with physical access to a sensor or actuator device it is possible to directly perform attacks equivalent of eavesdropping, manipulating data or denial of service. For example:

- Instead of eavesdropping the sensor data or attacking the authorization system to gain access to the data, the attacker could make its own measurements on the physical object.

- Instead of manipulating the sensor data the attacker could change the physical object which the sensor is measuring, thereby changing the payload data which is being sent.

- Instead of manipulating data for an actuator or attacking the authorization system, the attacker could perform an unauthorized action directly on the physical object.

- A denial-of-service attack could be performed physically on the object or device.

  All these attacks are possible by having physical access to the device, since the assets are related to the physical world. Moreover, this kind of attacks are in many cases straightforward (requires no special competence or tools, low cost given physical access, etc.)

As a conclusion, if an attacker has full physical access to a sensor or actuator device, then much of the security functionality elaborated in this draft is not effective to protect the asset during the physical attack.

Since it does not make sense to design a solution for a situation that cannot be protected against we assume there is no need to protect assets which are exposed during a physical attack. In other words, either an attacker does not have physical access to the sensor or actuator device, or if it has, the attack shall only have effect during the period of physical attack, and shall be limited in extent to the physical control the attacker exerts (e.g., must not affect the security of other devices.)

9.2. Time Measurements

Measuring time with certain accuracy is important to achieve certain security properties, for example to determine whether a public key certificate, access token or some other assertion is valid.

Dynamic authorization in itself requires the ability to handle expiry or revocation of authorization decisions or to distinguish new authorization decisions from old.

For certain categories of devices we can assume that there is an internal clock which is sufficiently accurate to handle the time measurement requirements. If RS can connect directly to AS it could get updated in terms of time as well as revocation information.

If RS continuously measures time but can’t connect to AS or other trusted source, time drift may have to be accepted and it may not be able to manage revocation. However, it may still be able to handle short lived access rights within some margins, by measuring the time since arrival of authorization information or request.

Some categories of devices in scope may be unable measure time with any accuracy (e.g. because of sleep cycles). This category of devices is not suitable for the use cases which require measuring validity of assertions and authorizations in terms of absolute time.

10. IANA Considerations

This document has no actions for IANA.

11. Acknowledgements

The authors would like to thank Olaf Bergmann, Robert Cragie, Klaus Hartke, Sandeep Kumar, John Mattson, Corinna Schmitt, Mohit Sethi, Hannes Tschofenig, Vlasios Tsiatsis and Erik Wahlstroem for contributing to the discussion, giving helpful input and commenting on previous forms of this draft. The authors would also like to specifically acknowledge input provided by Hummen and others [HUM14delegation].

12. Informative References

[HUM14delegation]
Hummen, R., Shafagh, H., Raza, S., Voigt, T., and K. Wehrle, "Delegation-based Authentication and Authorization for the IP-based Internet of Things", 11th IEEE International Conference on Sensing, Communication, and Networking (SECON’14), June 30 - July 3, 2014.

[I-D.garcia-core-security]
Garcia-Morchon, O., Kumar, S., Keoh, S., Hummen, R., and R. Struik, "Security Considerations in the IP-based Internet of Things", draft-garcia-core-security-06 (work in progress), September 2013.

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-13 (work in progress), April 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-03 (work in progress), March 2015.

[I-D.koster-core-coap-pubsub]
Koster, M., Keranen, A., and J. Jimenez, "Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)", draft-koster-core-coap-pubsub-01 (work in progress), March 2015.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., and L. Seitz, "March 9, 2015", draft-selander-ace-object-security-01 (work in progress), March 2015.

[OSCAR] Vucinic, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things", CoRR vol. abs/1404.7799, 2014.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

Authors' Addresses

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Ludwig Seitz
SICS Swedish ICT AB
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI
October 19, 2015

Delegated CoAP Authentication and Authorization Framework (DCAF) draft-gerdes-ace-dcaf-authorize-04

Abstract

This specification defines a protocol for delegating client authentication and authorization in a constrained environment for establishing a Datagram Transport Layer Security (DTLS) channel between resource-constrained nodes. The protocol relies on DTLS to transfer authorization information and shared secrets for symmetric cryptography between entities in a constrained network. A resource-constrained node can use this protocol to delegate authentication of communication peers and management of authorization information to a trusted host with less severe limitations regarding processing power and memory.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a transfer protocol similar to HTTP which is designed for the special requirements of constrained environments. A serious problem with constrained devices is the realization of secure communication. The devices only have limited system resources such as memory, stable storage (such as disk space) and transmission capacity and often lack input/output devices such as keyboards or displays. Therefore, they are not readily capable of using common protocols. Especially authentication mechanisms are difficult to realize, because the lack of stable storage severely limits the number of keys the system can store. Moreover, CoAP has no mechanism for authorization.

[I-D.ietf-ace-actors] describes an architecture that is designed to help constrained nodes with authorization-related tasks by introducing less-constrained nodes. These Authorization Managers perform complex security tasks for their nodes such as managing keys for numerous devices, and enable the constrained nodes to enforce the authorization policies of their principals.

DCAF uses access tokens to implement this architecture. A device that wants to access an item of interest on a constrained node first has to gain permission in the form of a token from the node's Authorization Manager.

As fine-grained authorization is not always needed on constrained devices, DCAF supports an implicit authorization mode where no authorization information is exchanged.

The main goals of DCAF are the setup of a Datagram Transport Layer Security (DTLS) [RFC6347] channel with symmetric pre-shared keys (PSK) [RFC4279] between two nodes and to securely transmit authorization tickets.

1.1. Features

- Utilize DTLS communication with pre-shared keys.
- Authenticated exchange of authorization information.
- Simplified authentication on constrained nodes by handing the more sophisticated authentication over to less-constrained devices.
- Support of secure constrained device to constrained device communication.
- Authorization policies of the principals of both participating parties are ensured.
- Simplified authorization mechanism for cases where implicit authorization is sufficient.
- Using only symmetric encryption on constrained nodes.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Readers are expected to be familiar with the terms and concepts defined in [I-D.ietf-ace-actors].

1.2.1. Actors

Server (S): An endpoint that hosts and represents a CoAP resource.

Client (C): An endpoint that attempts to access a CoAP resource on the Server.

Server Authorization Manager (SAM): An entity that prepares and endorses authentication and authorization data for a Server.

Client Authorization Manager (CAM): An entity that prepares and endorses authentication and authorization data for a Client.

Authorization Manager (AM): An entity that is either a SAM or a CAM.

Client Overseeing Principal (COP): The principal that is in charge of the Client and controls permissions concerning authorized representations of a CoAP resource.

Resource Overseeing Principal (ROP): The principal that is in charge of the CoAP resource and controls its access permissions.

1.2.2. Other Terms

Resource (R): A CoAP resource.

Authorization information: Contains all information needed by S to decide if C is privileged to access a resource in a specific way.

Authentication information: Contains all information needed by S to decide if the entity in possession of a certain key is verified by SAM.

Access information: Contains authentication information and, if necessary, authorization information.

Access ticket: Contains the authentication and, if necessary, the authorization information needed to access a resource. A Ticket consists of the Ticket Face and the Client Information. The access ticket is a representation of the access information.

Ticket Face: The part of the ticket which is generated for the Server. It contains the authorization information and all information needed by the Server to verify that it was granted by SAM.

Client Information (CI): The part of the ticket which is generated for the Client. It contains the Verifier and optionally may contain authorization information that represent COP’s authorization policies for C.

Client Authorization Information (CAI): A data structure that describes the C’s permissions for S according to CAM, e.g., which actions C is allowed to perform on an R of S.

Server Authorization Information (SAI): A data structure that describes C’s permissions for S according to SAM, e.g., which actions C is allowed to perform on an R of S.

Verifier: The secret (e.g. a 128-bit PSK) shared between C and S. It enables C to validate that it is communicating with a certain S and vice versa.

Explicit authorization: SAM informs the S in detail which privileges are granted to the Client.

Implicit authorization: SAM authenticates the Client for the Server without specifying the privileges in detail. This can be used for flat or unrestricted authorization (cf section 4 of [I-D.ietf-ace-actors]).

2. System Overview

Within the DCAF Architecture each Server (S) has a Server Authorization Manger (SAM) which conducts the authentication and authorization for S. S and SAM share a symmetric key which has to be exchanged initially to provide for a secure channel. The mechanism used for this is not in the scope of this document.

To gain access to a specific resource on a S, a Client (C) has to request an access ticket from the SAM serving S either directly or, if it is a constrained device, using its Client Authorization Manager (CAM). In the following, we always discuss the CAM role separately, even if that is co-located within a (more powerful) C (see section Section 11 for details about co-located actors).

CAM decides if S is an authorized source for R according to the policies set by COP and in this case transmits the request to SAM. If SAM decides that C is allowed to access the resource according to the policies set by ROP, it generates a DTLS pre-shared key (PSK) for the communication between C and S and wraps it into an access ticket. For explicit access control, SAM adds the detailed access permissions to the ticket in a way that CAM and S can interpret. CAM checks if the permissions in the access ticket comply with COP’s authorization policies for C, and if this is the case sends it to C. After C presented the ticket to S, C and S can communicate securely.

To be able to provide for the authentication and authorization services, an Authorization Manager has to fulfill several requirements:

- AM must have enough stable storage (such as disk space) to store the necessary number of credentials (matching the number of Clients and Servers).
- AM must possess means for user interaction, for example directly or indirectly connected input/output devices such as keyboard and

display, to allow for configuration of authorization information by the respective Principal.

- o AM must have enough processing power to handle the authorization requests for all constrained devices it is responsible for.

3. Protocol

The DCAF protocol comprises three parts:

1. transfer of authentication and, if necessary, authorization information between C and S;
2. transfer of access requests and the respective ticket transfer between C and CAM; and
3. transfer of ticket requests and the respective ticket grants between SAM and CAM.

3.1. Overview

In Figure 1, a DCAF protocol flow is depicted (messages in square brackets are optional):

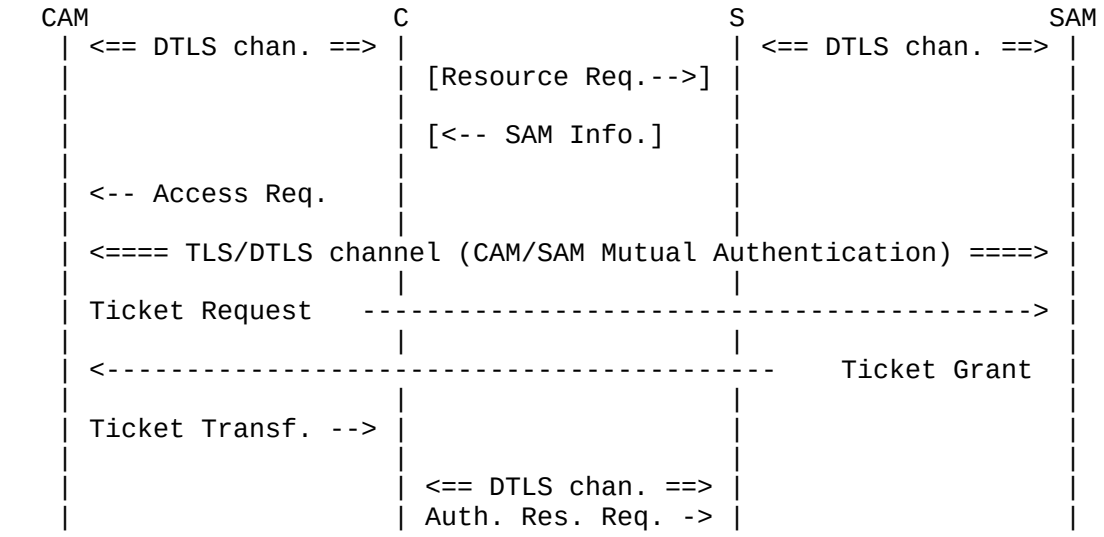


Figure 1: Protocol Overview

To determine the SAM in charge of a resource hosted at the S, C MAY send an initial Unauthorized Resource Request message to S. S then denies the request and sends the address of its SAM back to C.

Instead of the initial Unauthorized Resource Request message, C MAY look up the desired resource in a resource directory (cf. [I-D.ietf-core-resource-directory]) that lists S's resources as discussed in Section 9.

Once C knows SAM's address, it can send a request for authorization to SAM using its own CAM. CAM and SAM authenticate each other and each determine if the request is to be authorized. If it is, SAM generates an access ticket for C. The ticket contains keying material for the establishment of a secure channel and, if necessary, a representation of the permissions C has for the resource. C keeps one part of the access ticket and presents the other part to S to prove its right to access. With their respective parts of the ticket, C and S are able to establish a secure channel.

The following sections specify how CoAP is used to interchange access-related data between S and SAM so that SAM can provide C and S with sufficient information to establish a secure channel, and simultaneously convey authorization information specific for this communication relationship to S.

Note: Special implementation considerations apply when one single entity takes the role of more than one actors. Section 11 gives additional advice on some of these usage scenarios.

This document uses Concise Binary Object Representation (CBOR, [RFC7049]) to express authorization information as set of attributes passed in CoAP payloads. Notation and encoding options are discussed in Section 5. A formal specification of the DCAF message format is given in Appendix A.

3.2. Unauthorized Resource Request Message

The optional Unauthorized Resource Request message is a request for a resource hosted by S for which no proper authorization is granted. S MUST treat any CoAP request as Unauthorized Resource Request message when any of the following holds:

- The request has been received on an unprotected channel.
- S has no valid access ticket for the sender of the request regarding the requested action on that resource.

- o S has a valid access ticket for the sender of the request, but this does not allow the requested action on the requested resource.

Note: These conditions ensure that S can handle requests autonomously once access was granted and a secure channel has been established between C and S.

Unauthorized Resource Request messages MUST be denied with a client error response. In this response, the Server MUST provide proper SAM Information to enable the Client to request an access ticket from S's SAM as described in Section 3.3.

The response code MUST be 4.01 (Unauthorized) in case the sender of the Unauthorized Resource Request message is not authenticated, or if S has no valid access ticket for C. If S has an access ticket for C but not for the resource that C has requested, S MUST reject the request with a 4.03 (Forbidden). If S has an access ticket for C but it does not cover the action C requested on the resource, S MUST reject the request with a 4.05 (Method Not Allowed).

Note: The use of the response codes 4.03 and 4.05 is intended to prevent infinite loops where a dumb Client optimistically tries to access a requested resource with any access token received from the SAM. As malicious clients could pretend to be C to determine C's privileges, these detailed response codes must be used only when a certain level of security is already available which can be achieved only when the Client is authenticated.

3.3. SAM Information Message

The SAM Information Message is sent by S as a response to an Unauthorized Resource Request message (see Section 3.2) to point the sender of the Unauthorized Resource Request message to S's SAM. The SAM information is a set of attributes containing an absolute URI (see Section 4.3 of [RFC3986]) that specifies the SAM in charge of S.

An optional field A lists the different content formats that are supported by S.

The message MAY also contain a timestamp generated by S.

Figure 2 shows an example for an SAM Information message payload using CBOR diagnostic notation. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://sam.example.com/authorize", TS: 168537,
 A: [ TBD1, ct_cose_msg ] }
```

Figure 2: SAM Information Payload Example

In this example, the attribute SAM points the receiver of this message to the URI "coaps://sam.example.com/authorize" to request access permissions. The originator of the SAM Information payload (i.e. S) uses a local clock that is loosely synchronized with a time scale common between S and SAM (e.g., wall clock time). Therefore, it has included a time stamp on its own time scale that is used as a nonce for replay attack prevention. Refer to Section 4.1 for more details concerning the usage of time stamps to ensure freshness of access tickets.

The content formats accepted by S are TBD1 (identifying 'application/dcaf+cbor' as defined in this document), and 'application/cose+cbor' defined in [I-D.ietf-cose-msg].

Editorial note: ct_cose_msg is to be replaced with the numeric value assigned for 'application/cose+cbor'.

The examples in this document are written in CBOR diagnostic notation to improve readability. Figure 3 illustrates the binary encoding of the message payload shown in Figure 2.

```
a2                                  # map(2)
    00                              # unsigned(0) (=SAM)
    78 21                           # text(33)
       636f6170733a2f2f73616d2e6578
       616d706c652e636f6d2f617574686f72
       697a65             # "coaps://sam.example.com/authorize"
    05                              # unsigned(5) (=TS)
    1a 00029259                     # unsigned(168537)
    0a                              # unsigned(10) (=A)
    82                              # array(2)
       19 03e6                      # unsigned(998) (=dcaf+cbor)
       19 03e7                      # unsigned(999) (=cose+cbor)
```

Figure 3: SAM Information Payload Example encoded in CBOR

3.3.1. Piggybacked Protected Content

For some use cases (such as sleepy nodes) it might be necessary to store sensor data on a server that might not belong to the same security domain. A client can retrieve the data from that server.

To be able to achieve the security objectives of the principles the data must be protected properly.

The server that hosts the stored data may respond to GET requests for this particular resource with a SAM Information message that contains the protected data as piggybacked content. As the server may frequently publish updates to the stored data, the URI of the authorization manager responsible for the protected data MAY be omitted and must be retrieved from a resource directory.

Once a requesting client has received the SAM Information Message with piggybacked content, it needs to request authorization for accessing the protected data. To do so, it constructs an Access Request as defined in Section 3.4. If access to the protected data is granted, the requesting client will be provided with cryptographic material to verify the integrity and authenticity of the piggybacked content and decrypt the protected data in case it is encrypted.

3.4. Access Request

To retrieve an access ticket for the resource that C wants to access, C sends an Access Request to its CAM. The Access Request is constructed as follows:

1. The request method is POST.

2. The request URI is set as described below.

3. The message payload contains a data structure that describes the action and resource for which C requests an access ticket.

The request URI identifies a resource at CAM for handling authorization requests from C. The URI SHOULD be announced by CAM in its resource directory as described in Section 9.

Note: Where capacity limitations of C do not allow for resource directory lookups, the request URI in Access Requests could be hard-coded during provisioning or set in a specific device configuration profile.

The message payload is constructed from the SAM information that S has returned in its SAM Information message (see Section 3.3) and information that C provides to describe its intended request(s). The Access Request MUST contain the following attributes:

1. Contact information for the SAM to use.

2. An absolute URI of the resource that C wants to access.

3. The actions that C wants to perform on the resource.

4. Any time stamp generated by S.

An example Access Request from C to CAM is depicted in Figure 4. (Refer to Section 5 for a detailed description of the available attributes and their semantics.)

```
POST client-authorize
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://sam.example.com/authorize",
  SAI: ["coaps://temp451.example.com/s/tempC", 5],
  TS: 168537
}
```

Figure 4: Access Request Message Example

The example shows an Access Request message payload for the resource "/s/tempC" on the Server "temp451.example.com". Requested operations in attribute SAI are GET and PUT.

The attributes SAM (that denotes the Server Authorization Manager to use) and TS (a nonce generated by S) are taken from the SAM Information message from S.

The response to an Authorization Request is delivered by CAM back to C in a Ticket Transfer message.

3.5. Ticket Request Message

When CAM receives an Access Request message from C and COP specified authorization policies for C, CAM MUST check if the requested actions are allowed according to these policies. If all requested actions are forbidden, CAM MUST send a 4.03 response.

If no authorization policies were specified or some or all of the requested actions are allowed according to the authorization policies, CAM either returns a cached response or attempts to create a Ticket Request message. The Ticket Request message MAY contain all actions requested by C since CAM will add CAI in the Ticket Transfer Message if COP specified authorization policies (see Section 3.7).

CAM MAY return a cached response if it is known to be fresh according to Max-Age. CAM SHOULD NOT return a cached response if it expires in less than a minute.

If CAM does not send a cached response, it checks whether the request payload is of type "application/dcaf+cbor" and contains at least the fields SAM and SAI. CAM MUST respond with 4.00 (Bad Request) if the type is "application/dcaf+cbor" and any of these fields is missing or does not conform to the format described in Section 5.

If the payload is correct, CAM creates a Ticket Request message from the Access Request received from C as follows:

1. The destination of the Ticket Request message is derived from the "SAM" field that is specified in the Access Request message payload (for example, if the Access Request contained 'SAM: "coaps://sam.example.com/authz"', the destination of the Ticket Request message is sam.example.com).

2. The request method is POST.

3. The request URI is constructed from the SAM field received in the Access Request message payload.

4. The payload is copied from the Access Request sent by C.

To send the Ticket Request message to SAM a secure channel between CAM and SAM MUST be used. Depending on the URI scheme used in the SAM field of the Access Request message payload (the less-constrained devices CAM and SAM do not necessarily use CoAP to communicate with each other), this could be, e.g., a DTLS channel (for "coaps") or a TLS connection (for "https"). CAM and SAM MUST be able to mutually authenticate each other, e.g. based on a public key infrastructure. (Refer to Section 8 for a detailed discussion of the trust relationship between Client Authorization Managers and Server Authorization Managers.)

3.6. Ticket Grant Message

When SAM has received a Ticket Request message it has to evaluate the access request information contained therein. First, it checks whether the request payload is of type "application/dcaf+cbor" and contains at least the fields SAM and SAI. SAM MUST respond with 4.00 (Bad Request) for CoAP (or 400 for HTTP) if the type is "application/dcaf+cbor" and any of these fields is missing or does not conform to the format described in Section 5.

SAM decides whether or not access is granted to the requested resource and then creates a Ticket Grant message that reflects the result. To grant access to the requested resource, SAM creates an access ticket comprised of a Face and the Client Information as described in Section 4.

The Ticket Grant message then is constructed as a success response indicating attached content, i.e. 2.05 for CoAP, or 200 for HTTP, respectively. The payload of the Ticket Grant message is a data structure that contains the result of the access request. When access is granted, the data structure contains the Ticket Face and the Client Information. Face contains the SAI and the Session Key Generation Method. The CI at this point only consists of the Verifier.

The Ticket Grant message MAY provide cache-control options to enable intermediaries to cache the response. The message MAY be cached according to the rules defined in [RFC7252] to facilitate ticket retrieval when C has crashed and wants to recover the DTLS session with S.

SAM SHOULD set Max-Age according to the ticket lifetime in its response (Ticket Grant Message).

Figure 5 shows an example Ticket Grant message using CoAP. The Face/ Verifier information is transferred as a CBOR data structure as specified in Section 5. The Max-Age option tells the receiving CAM how long this ticket will be valid.

```
2.05 Content
Content-Format: application/dcaf+cbor
Max-Age: 86400
{ F: {
        SAI: [ "/s/tempC", 7 ],
        TS: 0("2013-07-10T10:04:12.391"),
        L:  86400,
        G: hmac_sha256
  },
  V: h'f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9'
}
```

Figure 5: Example Ticket Grant Message

A Ticket Grant message that declines any operation on the requested resource is illustrated in Figure 6. As no ticket needs to be issued, an empty payload is included with the response.

```
2.05 Content
Content-Format: application/dcaf+cbor
```

Figure 6: Example Ticket Grant Message With Reject

3.7. Ticket Transfer Message

A Ticket Transfer message delivers the access information sent by SAM in a Ticket Grant message to the requesting client C. The Ticket Transfer message is the response to the Access Request message sent from C to CAM and includes the ticket data from SAM contained in the Ticket Grant message.

The Authorization Information provided by SAM in the Ticket Grant Message may grant more permissions than C has requested. The authorization policies of COP and ROP may differ: COP might want restrict the resources C is allowed to access, and the actions that C is allowed to perform on the resource.

If COP defined authorization policies that concern the requested actions, CAM MUST add Authorization Information for C (CAI) to the CI that reflect those policies. Since C and CAM use a DTLS channel for communication, the autorization information does not need to be encrypted.

CAM includes the Face and the CI containing the verifier sent by SAM in the Ticket Transfer message. However, CAM MUST NOT include additional information SAM provided in CI. In particular, CAM MUST NOT include any CAI information provided by SAM, since CAI represents COP’s authorization policies that MUST NOT be provided by SAM.

Figure 7 shows an example Ticket Transfer message that conveys the permissions for actions GET, POST, PUT (but not DELETE) on the resource "/s/tempC" in field SAI. As CAM only wants to permit outbound GET requests, it restricts C’s permissions in the field CAI accordingly.

```
2.05 Content
Content-Format: application/dcaf+cbor
Max-Age: 86400
{ F: {
        SAI: [ "/s/tempC", 7 ],
        TS: 0("2013-07-10T10:04:12.391"),
        L:  86400,
        G: hmac_sha256
  },
  V: h'f89947160c73601c7a65cb5e08812026
       6d0f0565160e3ff7d3907441cdf44cc9'
  CAI: [ "/s/tempC", 1 ],
  TS: 0("2013-07-10T10:04:12.855"),
  L:  86400
}
```

Figure 7: Example Ticket Transfer Message

3.8. DTLS Channel Setup Between C and S

When C receives a Ticket Transfer message, it checks if the payload contains a face and a Client Information. With this information C can initiate establishment of a new DTLS channel with S. To use DTLS with pre-shared keys, C follows the PSK key exchange algorithm specified in Section 2 of [RFC4279], with the following additional requirements:

1. C sets the psk_identity field of the ClientKeyExchange message to the ticket Face received in the Ticket Transfer message.

2. C uses the ticket Verifier as PSK when constructing the premaster secret.

Note1: As S cannot provide C with a meaningful PSK identity hint in response to C's ClientHello message, S SHOULD NOT send a ServerKeyExchange message.

Note2: According to [RFC7252], CoAP implementations MUST support the ciphersuite TLS_PSK_WITH_AES_128_CCM_8 [RFC6655]. C is therefore expected to offer at least this ciphersuite to S.

Note3: The ticket is constructed by SAM such that S can derive the authorization information as well as the PSK (refer to Section 6 for details).

3.9. Authorized Resource Request Message

If the Client Information in the Ticket Transfer message contains CAI, C MUST ensure that it only sends requests that according to them are allowed. C therefore MUST check CAI, L and TS before every request. If CAI is no longer valid according to L, C MUST terminate the DTLS connection with S and re-request the CAI from CAM using an Access Request Message.

On the Server side, successful establishment of the DTLS channel between C and S ties the SAM authorization information contained in the psk_identity field to this channel. Any request that S receives on this channel is checked against these authorization rules. Incoming CoAP requests that are not Authorized Resource Requests MUST be rejected by S with 4.01 response as described in Section 3.2.

S SHOULD treat an incoming CoAP request as Authorized Resource Request if the following holds:

1. The message was received on a secure channel that has been established using the procedure defined in Section 3.8.

2. The authorization information tied to the secure channel is valid.

3. The request is destined for S.

4. The resource URI specified in the request is covered by the authorization information.

5. The request method is an authorized action on the resource with respect to the authorization information.

Note that the authorization information is not restricted to a single resource URI. For example, role-based authorization can be used to authorize a collection of semantically connected resources simultaneously. Implicit authorization also provides access rights to authenticated clients for all actions on all resources that S offers. As a result, C can use the same DTLS channel not only for subsequent requests for the same resource (e.g. for block-wise transfer as defined in [I-D.ietf-core-block] or refreshing observe-relationships [RFC7641]) but also for requests to distinct resources.

Incoming CoAP requests received on a secure channel according to the procedure defined in Section 3.8 MUST be rejected

1. with response code 4.03 (Forbidden) when the resource URI specified in the request is not covered by the authorization information, and

2. with response code 4.05 (Method Not Allowed) when the resource URI specified in the request covered by the authorization information but not the requested action.

Since SAM may limit the set of requested actions in its Ticket Grant message, C cannot know a priori if an Authorized Resource Request will succeed. If C repeatedly gets SAM Information messages as response to its requests, it SHOULD NOT send new Access Requests to CAM.

3.10. Dynamic Update of Authorization Information

Once a security association exists between a Client and a Resource Server, the Client can update the Authorization Information stored at the Server at any time. To do so, the Client creates a new Access Request for the intended action on the respective resource and sends this request to its CAM which checks and relays this request to the Server's SAM as described in Section 3.4.

Note: Requesting a new Access Ticket also can be a Client's reaction on a 4.03 or 4.05 error that it has received in response to an Authorized Resource Request.

Figure 8 depicts the message flow where C requests a new Access Tickets after a security association between C and S has been established using this protocol.

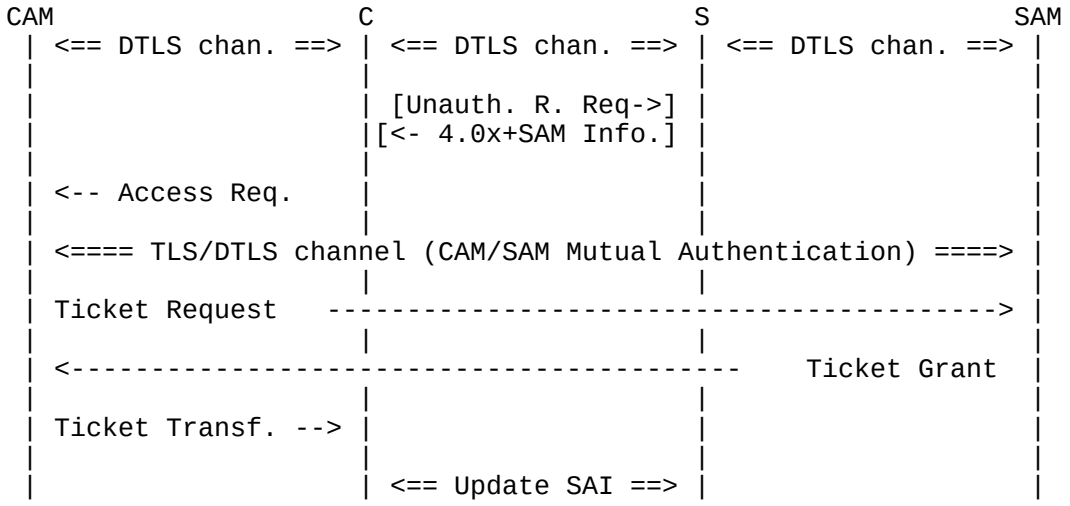


Figure 8: Overview of Dynamic Update Operation

Processing the Ticket Request is done at the SAM as specified in Section 3.6, i.e. the SAM checks whether or not the requested operation is permitted by the Resource Principal's policy, and then return a Ticket Grant message with the result of this check. If access is granted, the Ticket Grant message contains an Access Ticket comprised of a public Ticket Face and a private Ticket Verifier. This authorization payload is relayed by CAM to the Client in a Ticket Transfer Message as defined in Section 3.7.

The major difference between dynamic update of Authorization Information and the initial handshake is the handling of a Ticket Transfer message by the Client that is described in Section 3.10.1.

3.10.1. Handling of Ticket Transfer Messages

If the security association with S still exists and S has indicated support for session renegotiation according to [RFC5746], the ticket Face SHOULD be used to renegotiate the existing DTLS session. In this case, the ticket Face is used as psk_identity as defined in Section 3.8. Otherwise, the Client MUST perform a new DTLS handshake according to Section 3.8 that replaces the existing DTLS session.

After successful completion of the DTLS handshake S updates the existing SAM Authorization Information for C according to the contents of the ticket Face.

Note: No mutual authentication between C and S is required for dynamic updates when a DTLS channel exists that has been established as defined in Section 3.8. S only needs to verify the authenticity and integrity of the ticket Face issued by SAM which is achieved by having performed a successful DTLS handshake with the ticket Face as psk_identity. This could even be done within the existing DTLS session by tunneling a CoDTLS [I-D.schmertmann-dice-codtls] handshake.

4. Ticket

Access tokens in DCAF are tickets that consist of two parts, namely the Face and the Client Information (CI). SAM generates the ticket Face for S and the verifier that corresponds to the ticket Face for C. The verifier is included in the CI.

The Ticket is transmitted over CAM to C. C keeps the CI and sends the Face to S. CAM can add Client authorization information (CAI) for C to the CI if necessary.

S uses the information in the ticket Face to validate that it was generated by SAM and to authenticate and authorize the client. No additional information about the Client is needed, S keeps the Ticket Face as long as it is valid.

C uses the verifier to authenticate S. If CAM specified CAI, the client uses it to authorize the server.

The ticket is not required to contain a client or a server identifier. The ticket Face MAY contain an SAI identifier for revocation. The CI MAY contain a CAI identifier for revocation.

4.1. Face

Face is the part of the ticket that is generated by SAM for S. Face MUST contain all information needed for authorized access to a resource:

- SAM Authorization Information (SAI)
- A nonce

Optionally, Face MAY also contain:

- A lifetime (optional)
- A DTLS pre-shared key (optional)

- A SAI identifier (optional)

S MUST verify the integrity of Face, i.e. the information contained in Face stems from SAM and was not manipulated by anyone else. The integrity of Face can be ensured by various means. Face may be encrypted by SAM with a key it shares with S. Alternatively, S can use a mechanism to generate the DTLS PSK which includes Face. S generates the key from the Face it received. The correct key can only be calculated with the correct Face (refer to Section 6 for details).

Face MUST contain a nonce to verify that the contained information is fresh. As constrained devices may not have a clock, nonces MAY be generated using the clock ticks since the last reboot. To circumvent synchronization problems the timestamp MAY be generated by S and included in the first SAM Information message. Alternatively, SAM MAY generate the timestamp for the nonce. In this case, SAM and S MUST use a time synchronization mechanism to make sure that S interprets the timestamp correctly.

Face MAY contain an SAI identifier that uniquely identifies the SAI for S and SAM and can be used for revocation.

Face MAY be encrypted. If Face contains a DTLS PSK, the whole content of Face MUST be encrypted.

The ticket Face does not need to contain a client identifier.

4.2. Client Information

The CI part of the ticket is generated for C. It contains

- The Verifier generated by SAM

CI MAY additionally contain:

- CAI generated by CAM
- A nonce generated by CAM
- A lifetime generated by CAM
- A SAI identifier generated by CAM

CI MUST contain the verifier, i.e. the DTLS PSK for C. The Verifier MUST NOT be transmitted over unprotected channels.

Additionally, CI MAY contain CAI to provide the COP's authorization policies to C. If the CI contains CAI, CAM MUST add a nonce that enables C to validate that the information is fresh. CAM MAY use a timestamp as the nonce (see Section 4.1). CAM SHOULD add a lifetime to CI to limit the lifetime of the CAI. CAM MAY additionally add a CAI identifier to CI for revocating the CAI. The CAI identifier MUST uniquely identify the CAI for C and CAM.

4.3. Revocation

The existence of access tickets SHOULD be limited in time to avoid stale tickets that waste resources on S and C. This can be achieved either by explicit Revocation Messages to invalidate a ticket or implicitly by attaching a lifetime to the ticket.

The SAI in the ticket Face and the CAI in the CI need to be protected separately. CAM decides about the validity of the CAI while SAM is in charge of the validity of SAI. To be able to revoke the CAI, CAM SHOULD include a CAI identifier in the CI. SAM SHOULD include a SAI identifier in FACE to be able to revocate the SAI.

4.4. Lifetime

SAI and CAI MAY each have lifetime. SAM is responsible for defining the SAI lifetime, CAM is responsible for the CAI lifetime. If SAM sets a lifetime for SAI, SAM and S MUST use a time synchronization method to ensure that S is able to interpret the lifetime correctly. S SHOULD end the DTLS connection to C if the lifetime of a ticket has run out and it MUST NOT accept new requests. S MUST NOT accept tickets with an invalid lifetime.

If CAM provides CAI in the CI part of the ticket, CAM MAY add a lifetime for this CAI. If CI contains a lifetime, CAM and C MUST use a time synchronization method to ensure that C is able to interpret the lifetime correctly. C SHOULD end the DTLS connection to S and MUST NOT send new requests if the CAI in the ticket is no longer valid. C MUST NOT accept tickets with an invalid lifetime.

Note: Defining reasonable ticket lifetimes is difficult to accomplish. How long a client needs to access a resource depends heavily on the application scenario and may be difficult to decide for SAM.

4.4.1. Revocation Messages

SAM MAY revoke tickets by sending a ticket revocation message to S. If S receives a ticket revocation message, it MUST end the DTLS connection to C and MUST NOT accept any further requests from C.

If ticket revocation messages are used, S MUST check regularly if SAM is still available. If S cannot contact SAM, it MUST end all DTLS connections and reject any further requests from C.

Likewise, CAM MAY revoke tickets by sending a ticket revocation message to C. If C receives a CAI revocation message, it MUST end the DTLS connection to S and MUST NOT send any further requests to S.

If CAI revocation messages are used, C MUST check regularly if CAM is still available. If C cannot contact CAM, it MUST end all DTLS connections and MUST NOT send any more requests to S.

Note: The loss of the connection between S and SAM prevents all access to S. This might especially be a severe problem if SAM is responsible for several Servers or even a whole network.

5. Payload Format and Encoding (application/dcaf+cbor)

Various messages types of the DCAF protocol carry payloads to express authorization information and parameters for generating the DTLS PSK to be used by C and S. In this section, a representation in Concise Binary Object Representation (CBOR, [RFC7049]) is defined.

DCAF data structures are defined as CBOR maps that contain key value pairs. For efficient encoding, the keys defined in this document are represented as unsigned integers in CBOR, i. e. major type 0. For improved reading, we use symbolic identifiers to represent the corresponding encoded values as defined in Table 1.

Encoded Value	Key
0	SAM
1	SAI
2	CAI
3	E
4	K
5	TS
6	L
7	G
8	F
9	V
10	A
11	D
12	N

Table 1: DCAF field identifiers encoded in CBOR

The following list describes the semantics of the keys defined in DCAF.

SAM: Server Authorization Manager. This attribute denotes the Server Authorization Manager that is in charge of the resource specified in attribute R. The attribute's value is a string that contains an absolute URI according to Section 4.3 of [RFC3986].

SAI: SAM Authorization Information. A data structure used to convey authorization information from SAM to S. It describes C's permissions for S according to SAM, e.g., which actions C is allowed to perform on an R of S. The SAI attribute contains an AIF object as defined in [I-D.bormann-core-ace-aif]. C uses SAI for its Access Request messages.

CAI: CAM Authorization Information. A data structure used to convey authorization information from CAM to C. It describes the C's permissions for S according to CAM, e.g., which actions C is allowed to perform on an R of S. The CAI attribute contains an AIF object as defined in [I-D.bormann-core-ace-aif].

A: Accepted content formats. An array of numeric content formats from the CoAP Content-Formats registry (c.f. Section 12.3 of [RFC7252].

D: Protected Data. A binary string containing data that may be encrypted.

E: Encrypted Ticket Face. A binary string containing an encrypted ticket Face.

K: Key. A string that identifies the shared key between S and SAM that can be used to decrypt the contents of E. If the attribute E is present and no attribute K has been specified, the default is to use the current session key for the secured channel between S and SAM.

TS: Time Stamp. A time stamp that indicates the instant when the access ticket request was formed. This attribute can be used by the Server in an SAM Information message to convey a time stamp in its local time scale (e.g. when it does not have a real time clock with synchronized global time). When the attribute's value is encoded as a string, it MUST contain a valid UTC timestamp without time zone information. When encoded as integer, TS contains a system timestamp relative to the local time scale of its generator, usually S.

L: Lifetime. When in included in a ticket face, the contents of the L parameter denote the lifetime of the ticket. In combination with the protected data field D, this parameter denotes the lifetime of the protected data. When encoded as a string, L MUST denote the ticket's expiry time as a valid UTC timestamp without time zone information. When encoded as an integer, L MUST denote the ticket's validity period in seconds relative to TS.

N: Nonce. An initialization vector used in combination with piggybacked protected content.

G: DTLS PSK Generation Method. A numeric identifier for the method that S MUST use to derive the DTLS PSK from the ticket Face. This attribute MUST NOT be used when attribute V is present within the contents of F. This specification uses symbolic identifiers for improved readability. The corresponding numeric values encoded in

CBOR are defined in Table 2. A registry for these codes is defined in Section 13.1.

F: Ticket Face. An object containing the fields SAI, TS, and optionally G, L and V.

V: Ticket Verifier. A binary string containing the shared secret between C and S.

Encoded Value	Mnemonic	Support
0	hmac_sha256	mandatory
1	hmac_sha384	optional
2	hmac_sha512	optional

Table 2: CBOR encoding for DTLS PSK Key Generation Methods

5.1. Examples

The following example specifies a SAM that will be accessed using HTTP over TLS. The request URI is set to "/a?ep=%5B2001:DB8::dcaf:1234%5D" (hence denoting the endpoint address to authorize). TS denotes a local timestamp in UTC.

```
POST /a?ep=%5B2001:DB8::dcaf:1234%5D HTTP/1.1
Host: sam.example.com
Content-Type: application/dcaf+cbor
{SAM: "https://sam.example.com/a?ep=%5B2001:DB8::dcaf:1234%5D",
 SAI: ["coaps://temp451.example.com/s/tempC", 1],
 TS: 0("2013-07-14T11:58:22.923")}
```

The following example shows a ticket for the distributed key generation method (cf. Section 6.2), comprised of a Face (F) and a Verifier (V). The Face data structure contains authorization information SAI, a client descriptor, a timestamp using the local time scale of S, and a lifetime relative to S’s time scale.

The DTLS PSK Generation Method is set to hmac_sha256 denoting that the distributed key derivation is used as defined in Section 6.2 with SHA-256 as HMAC function.

The Verifier V contains a shared secret to be used as DTLS PSK between C and S.

```
HTTP/1.1 200 OK
Content-Type: application/dcaf+cbor
{
  F: {
       SAI: [ "/s/tempC", 1 ],
       TS: 2938749,
       L:  3600,
       G: hmac_sha256
     },
  V: h'48ae5a81b87241d81618f56cab0b65ec
       441202f81faabbe10075b20cb57fa939'
}
```

The Face may be encrypted as illustrated in the following example. Here, the field E carries an encrypted Face data structure that contains the same information as the previous example, and an additional Verifier. Encryption was done with a secret shared by SAM and S. (This example uses AES128_CCM with the secret { 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f } and S's timestamp { 0x00, 0x2C, 0xD7, 0x7D } as nonce.) Line breaks have been inserted to improve readability.

The attribute K describes the identity of the key to be used by S to decrypt the contents of attribute E. Here, The value "key0" in this example is used to indicate that the shared session key between S and SAM was used for encrypting E.

```
{
  E: h'2e75eeae01b831e0b65c2976e06d90f4
       82135bec5efef3be3d31520b2fa8c6fb
       f572f817203bf7a0940bb6183697567c
       e291b03e9fca5e9cbdfa7e560322d4ed
       3a659f44a542e55331a1a9f43d7f',
  K: "key0",
  V: h'48ae5a81b87241d81618f56cab0b65ec
       441202f81faabbe10075b20cb57fa939'
}
```

The decrypted contents of E are depicted below (whitespace has been added to improve readability). The presence of the attribute V indicates that the DTLS PSK Transfer is used to convey the session key (cf. Section 6.1).

```
{
  F: {
       SAI: [ "/s/tempC", 1 ],
       TS: 2938749,
       L:  3600,
       G: hmac_sha256
     },
  V: h'48ae5a81b87241d81618f56cab0b65ec
       441202f81faabbe10075b20cb57fa939'
}
```

6. DTLS PSK Generation Methods

One goal of the DCAF protocol is to provide for a DTLS PSK shared between C and S. SAM and S MUST negotiate the method for the DTLS PSK generation.

6.1. DTLS PSK Transfer

The DTLS PSK is generated by AS and transmitted to C and S using a secure channel.

The DTLS PSK transfer method is defined as follows:

- SAM generates the DTLS PSK using an algorithm of its choice
- SAM MUST include a representation of the DTLS PSK in Face and encrypt it together with all other information in Face with a key K(SAM,S) it shares with S. How SAM and S exchange K(SAM,S) is not in the scope of this document. SAM and S MAY use their preshared key as K(SAM,S).
- SAM MUST include a representation of the DTLS PSK in the Verifier.
- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.
- S MUST decrypt Face using K(SAM,S)

6.2. Distributed Key Derivation

SAM generates a DTLS PSK for C which is transmitted using a secure channel. S generates its own version of the DTLS PSK using the information contained in Face (see also Section 4.1).

The distributed key derivation method is defined as follows:

- SAM and S both generate the DTLS PSK using the information included in Face. They use an HMAC algorithm on Face with a shared key K(SAM,S). The result serves as the DTLS PSK. How SAM and S exchange K(SAM,S) is not in the scope of this document. They MAY use their preshared key as K(SAM,S). How SAM and S negotiate the used HMAC algorithm is also not in the scope of this document. They MAY however use the HMAC algorithm they use for their DTLS connection.

- SAM MUST include a representation of the DTLS PSK in the Verifier.

- As SAM and C do not have a shared secret, the Verifier MUST be transmitted to C using encrypted channels.

- SAM MUST NOT include a representation of the DTLS PSK in Face.

- SAM MUST NOT encrypt Face.

7. Authorization Configuration

For the protocol defined in this document, proper configuration of CAM and SAM is crucial. The principals that are in charge of the resource, S and SAM, and the principals that are in charge of C and CAM need to define the respective permissions. The data representation of these permissions are not in the scope of this document.

8. Trust Relationships

The constrained devices may be too constrained to manage complex trust relationships. Thus, DCAF does not require the constrained devices to perform complex tasks such as identifying a formerly unknown party. Each constrained device has a trust relationship with its respective AM. These less constrained devices are able to perform the more complex security tasks and can establish security associations with formerly unknown parties. The AMs hand down these security associations to their respective constrained device. The constrained devices require the help of their AMs for authentication and authorization.

C has a trust relationship with CAM: C trusts CAM to act in behalf of COP. S has a trust relationship with SAM: S trusts SAM to act in behalf of ROP. CAM trusts C to handle the data according to the CAI. SAM trusts S to protect resources according to the SAI. How the trust relationships between AMs and their respective constrained devices are established, is not in the scope of this document. It may be achieved by using a bootstrapping mechanism similar to [bergmann12] or by the means introduced in [I-D.gerdes-ace-a2a].

Additionally, SAM and CAM need to have established a trust relationship. Its establishment is not in the scope of this document. It fulfills the following conditions:

1. SAM and CAM have means to mutually authenticate each other (e.g., they might have a certificate of the other party or a PKI in which it is included)

2. If SAM requires information about the client from SAM, e.g. if SAM only wans to authorize certain types of devices, it can be sure that CAM correctly identifies these clients towards SAM and does not leak tickets that have been generated for a specific client C to another client.

SAM trusts C indirectly because it trusts CAM and CAM vouches for C. The DCAF Protocol does not provide any means for SAM to validate that a resource request stems from a specific C.

C indirectly entrusts SAM with some potentially confidential information, and trusts that SAM correctly represents S, because CAM trusts SAM.

CAM trusts S indirectly because it trusts SAM and SAM vouches for S.

C implicitly entrusts S with some potentially confidential information and trusts it to correctly represent R because it trusts CAM and because S can prove that it shares a key with SAM.

9. Listing Authorization Manager Information in a Resource Directory

CoAP utilizes the Web Linking format [RFC5988] to facilitate discovery of services in an M2M environment. [RFC6690] defines specific link parameters that can be used to describe resources to be listed in a resource directory [I-D.ietf-core-resource-directory].

9.1. The "auth-request" Link Relation

This section defines a resource type "auth-request" that can be used by clients to retrieve the request URI for a server's authorization service. When used with the parameter rt in a web link, "auth-request" indicates that the corresponding target URI can be used in a POST message to request authorization for the resource and action that are described in the request payload.

The Content-Format "application/dcaf+cbor with numeric identifier TBD1 defined in this specification MAY be used to express access requests and their responses.

The following example shows the web link used by CAM in this document to relay incoming Authorization Request messages to SAM. (Whitespace is included only for readability.)

```
<client-authorize>;rt="auth-request";ct=TBD1
                  ;title="Contact Remote Authorization Manager"
```

The resource directory that hosts the resource descriptions of S could list the following description. In this example, the URI "ep/node138/a/switch2941" is relative to the resource context "coaps://sam.example.com/", i.e. the Server Authorization Manager SAM.

```
<ep/node138/a/switch2941>;rt="auth-request";ct=TBD1;ep="node138"
                         ;title="Request Client Authorization"
                         ;anchor="coaps://sam.example.com/"
```

10. Examples

This section gives a number of short examples with message flows for the initial Unauthorized Resource Request and the subsequent retrieval of a ticket from SAM. The notation here follows the actors conventions defined in Section 1.2.1. The payload format is encoded as proposed in Section 5. The IP address of SAM is 2001:DB8::1, the IP address of S is 2001:DB8::dcaf:1234, and C's IP address is 2001:DB8::c.

10.1. Access Granted

This example shows an Unauthorized PUT request from C to S that is answered with a SAM Information message. C then sends a POST request to CAM with a description of its intended request. CAM forwards this request to SAM using CoAP over a DTLS-secured channel. The response from SAM contains an access ticket that is relayed back to CAM.

```
C --> S
PUT a/switch2941 [Mid=1234]
Content-Format: application/senml+json
{"e": [{"bv": "1"}]}

C <-- S
4.01 Unauthorized  [Mid=1234]
Content-Format: application/dcaf+cbor
{SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941"}

C --> CAM
POST client-authorize [Mid=1235,Token="tok"]
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 4]
}

CAM --> SAM [Mid=23146]
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 4]
}

CAM <-- SAM
2.05 Content  [Mid=23146]
Content-Format: application/dcaf+cbor
{ F: {
       SAI: ["a/switch2941", 5],
       TS: 0("2013-07-04T20:17:38.002"),
       G: hmac_sha256
     },
  V: h'7ba4d9e287c8b69dd52fd3498fb8d26d
       9503611917b014ee6ec2a570d857987a'
}

C <-- CAM
2.05 Content  [Mid=1235,Token="tok"]
Content-Format: application/dcaf+cbor
{ F: {
       SAI: ["a/switch2941", 5],
       TS: 0("2013-07-04T20:17:38.002"),
       G: hmac_sha256
     },
  V: h'7ba4d9e287c8b69dd52fd3498fb8d26d
       9503611917b014ee6ec2a570d857987a'
```

```
   }

   C --> S
   ClientHello (TLS_PSK_WITH_AES_128_CCM_8)

   C <-- S
   ServerHello (TLS_PSK_WITH_AES_128_CCM_8)
   ServerHelloDone

   C --> S
   ClientKeyExchange
     psk_identity=0xa301826c612f73776974636832393431
                  0x0505c077323031332d30372d30345432
                  0x303a31373a33382e3030320700

   (C decodes the contents of V and uses the result as PSK)
   ChangeCipherSpec
   Finished

   (S calculates PSK from SAI, TS and its session key
      HMAC_sha256(0xa301826c612f73776974636832393431
                  0x0505c077323031332d30372d30345432
                  0x303a31373a33382e3030320700,
                  0x736563726574)
   = 0x7ba4d9e287c8...
   )

   C <-- S
   ChangeCipherSpec
   Finished
```

10.2. Access Denied

This example shows a denied Authorization request for the DELETE operation.

```
C --> S
DELETE a/switch2941

C <-- S
4.01 Unauthorized
Content-Format: application/dcaf+cbor
{SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941"}

C --> CAM
POST client-authorize
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 8]
}

CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 8]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor

C <-- CAM
2.05 Content
Content-Format: application/dcaf+cbor
```

10.3. Access Restricted

This example shows a denied Authorization request for the operations GET, PUT, and DELETE. SAM grants access for PUT only.

```
CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
  SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
  SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 13]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor
{ F: {
       SAI: ["a/switch2941", 5],
       TS: 0("2013-07-04T21:33:11.930"),
       G: hmac_sha256
     },
  V: h'c7b5774f2ddcbd548f4ad74b30a1b2e5
       b6b04e66a9995edd2545e5a06216c53d'
}
```

10.4. Implicit Authorization

This example shows an Authorization request using implicit authorization. CAM initially requests the actions GET and POST on the resource "coaps://[2001:DB8::dcaf:1234]/a/switch2941". SAM returns a ticket that has no SAI field in its ticket Face, hence implicitly authorizing C.

```
CAM --> SAM
POST ep/node138/a/switch2941
Content-Format: application/dcaf+cbor
{
   SAM: "coaps://[2001:DB8::1]/ep/node138/a/switch2941",
   SAI: ["coaps://[2001:DB8::dcaf:1234]/a/switch2941", 3]
}

CAM <-- SAM
2.05 Content
Content-Format: application/dcaf+cbor
{ F: {
       TS: 0("2013-07-16T10:15:43.663"),
       G: hmac_sha256
     },
  V: h'4f7b0e7fdcc498fb2ece648bf6bdf736
       61a6067e51278a0078e5b8217147ea06'
}
```

11. Specific Usage Scenarios

The general DCAF architure outlined in Section 3.1 illustrates the various actors who participate in the message exchange for authenticated authorization. The message types defined in this document cover the most general case where all four actors are separate entities that may or may not reside on the same device.

Special implementation considerations apply when one single entity takes the role of more than one actor. This section gives advice on the most common usage scenarios where the Client Authorization Manager and Client, the Server Authorization Manager and Server or both Authorization Managers reside on the same (less-constrained) device and have a means of secure communication outside the scope of this document.

11.1. Combined Authorization Manager and Client

When CAM and C reside on the same (less-constrained) device, the Access Request and Ticket Transfer messages can be substituted by other means of secure communication. Figure 9 shows a simplified message exchange for a combined CAM+C device.

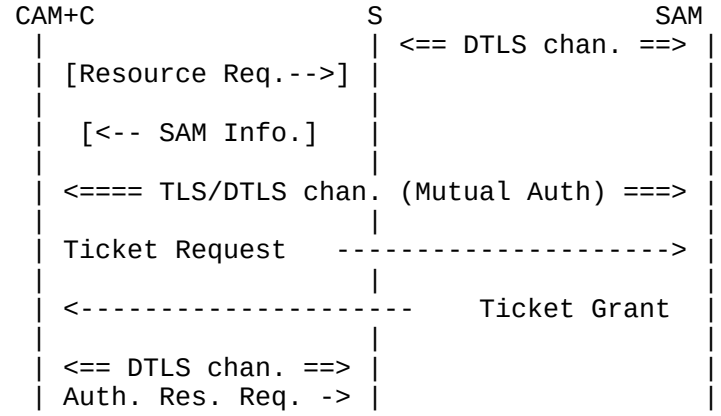


Figure 9: Combined Client Authorization Manager and Client

11.1.1. Creating the Ticket Request Message

When CAM+C receives an SAM Information message as a reaction to an Unauthorized Request message, it creates a Ticket Request message as follows:

1. The destination of the Ticket Request message is derived from the authority information in the URI contained in field "SAM" of the SAM Information message payload.

2. The request method is POST.

3. The request URI is constructed from the SAM field received in the SAM Information message payload.

4. The payload contains the SAM field from the SAM Information message, an absolute URI of the resource that CAM+C wants to access, the actions that CAM+C wants to perform on the resource, and any time stamp generated by S that was transferred with the SAM Information message.

11.1.2. Processing the Ticket Grant Message

Based on the Ticket Grant message, CAM+C is able to establish a DTLS channel with S. To do so, CAM+C sets the psk_identity field of the DTLS ClientKeyExchange message to the ticket Face received in the Ticket Grant message and uses the ticket Verifier as PSK when constructing the premaster secret.

11.2. Combined Client Authorization Manager and Server Authorization Manager

In certain scenarios, CAM and SAM may be combined to a single entity that knows both, C and S, and decides if their actions are authorized. Therefore, no explicit communication between CAM and SAM is necessary, resulting in omission of the Ticket Request and Ticket Grant messages. Figure 10 depicts the resulting message sequence in this simplified architecture.

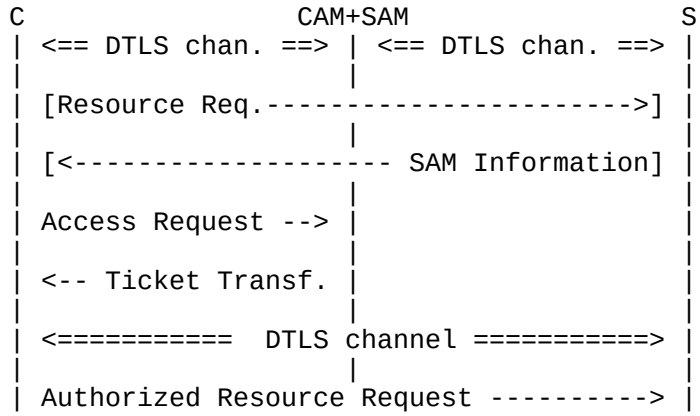


Figure 10: Combined Client Authorization Manager and Server Authorization Manager

11.2.1. Processing the Access Request Message

When receiving an Access Request message, CAM+SAM performs the checks specified in Section 3.5 and returns a 4.00 (Bad Request) response in case of failure. Otherwise, if the checks have succeeded, CAM+SAM evaluates the contents of Access Request message as described in Section 3.6.

The decision on the access request is performed by CAM+SAM with respect to the stored policies. When the requested action is permitted on the respective resource, CAM+SAM generates an access ticket as outlined in Section 4.1 and creates a Ticket Transfer message to convey the access ticket to the Client.

11.2.2. Creating the Ticket Transfer Message

A Ticket Transfer message is constructed as a 2.05 response with the access ticket contained in its payload. The response MAY contain a Max-Age option to indicate the ticket’s lifetime to the receiving Client.

This specification defines a CBOR data representation for the access ticket as illustrated in Section 3.6.

11.3. Combined Server Authorization Manager and Server

If SAM and S are colocated in one entity (SAM+S), the main objective is to allow CAM to delegate access to C. Accordingly, the authorization information could be replaced by a nonce internal to SAM+S. (TBD.)

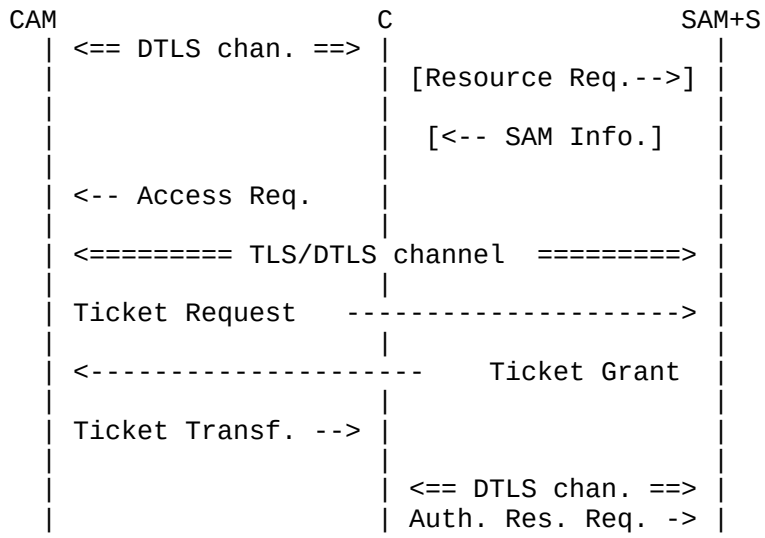


Figure 11: Combined Server Authorization Manager and Server

12. Security Considerations

As this protocol builds on transitive trust between Authorization Managers as mentioned in Section 8, SAM has no direct means to validate that a resource request originates from C. It has to trust CAM that it correctly vouches for C and that it does not give authorization tickets meant for C to another client nor disclose the contained session key.

The Authorization Managers also could constitute a single point of failure. If the Server Authorization Manager fails, the resources on all Servers it is responsible for cannot be accessed any more. If a Client Authorization Manager fails, all clients it is responsible are not able to access resources on a Server. Thus, it is crucial for large networks to use Authorization Managers in a redundant setup.

13. IANA Considerations

The following registrations are done following the procedure specified in [RFC6838].

Note to RFC Editor: Please replace all occurrences of "[RFC-XXXX]" with the RFC number of this specification.

13.1. DTLS PSK Key Generation Methods

A sub-registry for the values indicating the PSK key generation method as contents of the field G in a payload of type application/ dcaf+cbor is defined. Values in this sub-registry are numeric integers encoded in Concise Binary Object Notation (CBOR, [RFC7049]). This document follows the notation of [RFC7049] for binary values, i.e. a number starts with the prefix "0b". The major type is separated from the actual numeric value by an underscore to emphasize the value's internal structure.

Initial entries in this sub-registry are as follows:

Encoded Value	Name	Reference
0b000_00000	hmac_sha256	[RFC-XXXX]
0b000_00001	hmac_sha384	[RFC-XXXX]
0b000_00010	hmac_sha512	[RFC-XXXX]

Table 3: DTLS PSK Key Generation Methods

New methods can be added to this registry based on designated expert review according to [RFC5226].

(TBD: criteria for expert review.)

13.2. dcaf+cbor Media Type Registration

Type name: application

Subtype name: dcaf+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC7049]. Specifically, only the primitive data types String and Number are allowed. The type Number is restricted to unsigned integers (i.e., no negative numbers, fractions or exponents are allowed). Encoding MUST be UTF-8. These restrictions simplify implementations on devices that have very limited memory capacity.

Security considerations: TBD

Interoperability considerations: TBD

Published specification: [RFC-XXXX]

Applications that use this media type: TBD

Additional information:

Magic number(s): none

File extension(s): dcaf

Macintosh file type code(s): none

Person & email address to contact for further information: TBD

Intended usage: COMMON

Restrictions on usage: None

Author: TBD

Change controller: IESG

13.3. CoAP Content Format Registration

This document specifies a new media type application/dcaf+cbor (cf. Section 13.2). For use with CoAP, a numeric Content-Format identifier is to be registered in the "CoAP Content-Formats" sub-registry within the "CoRE Parameters" registry.

Note to RFC Editor: Please replace all occurrences of "RFC-XXXX" with the RFC number of this specification.

Media type	Encoding	Id.	Reference
application/dcaf+cbor	-	TBD1	[RFC-XXXX]

14. Acknowledgements

The authors would like to thank Renzo Navas for his valuable input and feedback.

15. References

15.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

15.2. Informative References

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-03 (work in progress), July 2015.

[I-D.gerdes-ace-a2a]
Gerdes, S., "Managing the Authorization to Authorize in the Lifecycle of a Constrained Device", draft-gerdes-ace-a2a-01 (work in progress), September 2015.

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-07 (work in progress), October 2015.

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-02 (work in progress), October 2015.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[I-D.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-05 (work in progress), October 2015.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-ietf-cose-msg-06 (work in progress), October 2015.

[I-D.schmertmann-dice-codtls]
Schmertmann, L., Hartke, K., and C. Bormann, "CoDTLS: DTLS handshakes over CoAP", draft-schmertmann-dice-codtls-01 (work in progress), August 2014.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <http://www.rfc-editor.org/info/rfc6655>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

[bergmann12] Bergmann, O., Gerdes, S., Schaefer, S., Junge, F., and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network", IEEE Wireless Communications and Networking Conference Workshops (WCNCW), April 2012.

Appendix A. CDDL Specification

This appendix shows a formal specification of the DCAF messaging format using the CBOR data definition language (CDDL) [I-D.greevenbosch-appsawg-cbor-cddl]:

```
dcaf-msg = sam-information-msg
         / access-request-msg
         / ticket-transfer-msg
         / ticket-grant-msg

sam-information-msg = { sam, ? full-timestamp, ? accepted-formats,
                        ? piggybacked }

access-request-msg = { sam, sam-ai, full-timestamp }

ticket-transfer-msg = { face-or-encrypted, verifier }
face-or-encrypted = ( face | encrypted-face )
face = ( F => { sam-ai, limited-timestamp, lifetime, psk-gen } )
verifier = ( V => shared-secret )
shared-secret = bstr
F   = 8
V   = 9

encrypted-face = ( E => bstr, K => tstr )
E   = 3
K   = 4
```

```
   ticket-grant-msg    = { face-or-encrypted, verifier, ? client-info }
   client-info = ( cam-ai, full-timestamp, lifetime)

   sam = (SAM => abs-uri)
   SAM = 0
   abs-uri = tstr ; .regexp "_____"

   sam-ai = ( SAI => [* auth-info])
   SAI = 1
   auth-info = ( uri : tstr, mask : 0..15 )

   cam-ai = ( CAI => [* auth-info])
   CAI = 2

   full-timestamp = ( TS => date)
   TS  = 5
   date = tdate / localdate
   localdate = uint
   limited-timestamp = ( TS => localdate)

   accepted-formats = ( A => [+ content-format] )
   content-format = uint ; valid entry from CoAP content format registry
   A=10

   piggybacked = ( data, lifetime, nonce )
   data = ( D => bstr )
   none = ( N => bstr )
   lifetime = ( L => period)
   period = uint ; in seconds
   L   = 6
   D   = 11
   N   = 12

   psk-gen = ( G => mac-algorithm)
   G   = 7
   mac-algorithm = &( hmac-sha256: 0, hmac-sha384: 1, hmac-sha512: 2 )
```

Authors' Addresses

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: January 7, 2016

S. Gerdes
O. Bergmann
C. Bormann
Universitaet Bremen TZI

July 06, 2015

Examples for Using DCAF with less constrained devices
draft-gerdes-ace-dcaf-examples-00

Abstract

Constrained nodes are devices which are limited in terms of processing power, memory, non-volatile storage and transmission capacity. Due to these constraints, commonly used security protocols are not easily applicable. Nevertheless, an authentication and authorization solution is needed to ensure the security of these devices.

The Delegated CoAP Authorization Framework (DCAF) specifies how resource-constrained nodes can delegate defined authentication- and authorization-related tasks to less-constrained devices called Authorization Managers, thus limiting the hardware requirements of the security solution for the constrained devices.

To realize the vision of "one Internet for all", constrained devices need to securely establish trust relationships with less constrained devices. This document lists examples for using DCAF with less constrained devices.

Table of Contents

1. Introduction

See abstract.

2. Terminology

- Readers should be familiar with the concepts introduced in [I-D.gerdes-ace-dcaf-authorize]

To reduce the confusion between OpenID and DCAF services, we are using the following terminology:

- Server Authorization Manager (SAM): An entity that prepares and endorses authentication and authorization data for a Server.
- Client Authorization Manager (CAM): An entity that prepares and endorses authentication and authorization data for a Client.
- Server (S): An endpoint that hosts and represents a resource.

- o Client (C): An endpoint that attempts to access a resource on the Server.

3. Example 1: Posting Sensor Values on a Website using OpenID Connect

To illustrate this example, we assume the following scenario: Sarah has a constrained device that is equipped with a temperature sensor. During a heat wave, she wants her device to continuously post the current room temperature in her office in the blog of her social media account to inform others of her working conditions. She wants to use OpenID Connect to log into CAM and establish a trust relationship between the temperature sensor and her blog.

The temperature sensor that is acting as a CoAP client (C) in this scenario is associated with a Client Authorization Manager (CAM) that helps C with authentication and authorization and provides a user interface to Sarah for configuring C. The blog on her social media account acts as a DCAF server (S).

In this first example, Sarah is registered with an OpenID provider (OP) that CAM accepts for authentication and which additionally acts as an Authorization Server (AS) for her social media service. Moreover, AS is compatible with DCAF and acts as a DCAF server authorization manager (SAM). For requesting access to the blog, OP defines the scope parameter coaps://blog.socialmedia.example.com. Sarah uses the browser on her smartphone as a User Agent (UA) to initiate the authentication and authorization process.

This example illustrates the authentication and authorization using the OpenID Connect Authorization Code Flow as described in section 3.1 of [OpenID-Connect].

Sarah uses her UA to request CAM to configure C to access her blog. To authenticate Sarah, CAM generates an Authentication Request (cf. section 3.1.2.1 of [OpenID-Connect]). Since CAM needs to obtain authorization for accessing Sarah’s Blog for C, it also adds the scope parameter coaps://blog.socialmedia.example.com. CAM sends the Authentication Request to Sarah’s UA, thereby triggering it to send an Authentication Request to OP’s Authorization Endpoint.

```
HTTP/1.1 302 Found
Location: https://server.example.com/authorize?
  response_type=code
  &scope=openid%20coaps://blog.socialmedia.example.com
  &client_id=s6BhdRkqt3
  &state=af0ifjsldkj
  &redirect_uri=https%3A%2F%2Fcam.example.org%2Flogin
```

After authenticating Sarah and requesting her permission (for details please consult sections 3.1.2.3 and 3.1.2.4 of [OpenID-Connect]), the OP sends back a successful Authentication Response that contains the parameter code. The code can then be used by CAM to send a Token Request to the Token Endpoint which responds with an ID token, an access token and a refresh token. CAM uses the ID token and access token to authenticate Sarah. The refresh token is then used to request an additional access token for accessing the blog.

CAM sends the refresh token to the Token Endpoint with the scope coaps://blog.socialmedia.example.com. The Token Endpoint validates the refresh token and makes sure that the requested scope does fit to the original grant. The Token Endpoint speaks both DCAF and OAuth and acts as SAM. It generates a ticket including a verifier and sends it back to CAM. CAM transmits the ticket to its client and instructs it to access the blog. The client then uses the ticket to establish a DTLS connection with the blog.

Figure 1 illustrates the resulting message flow.

(Please refer to PDF version to see this picture.)

Figure 1: Authentication and Authorization Flow

4. Security Considerations

TBD

5. IANA Considerations

None

6. References

6.1. Normative References

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02 (work in progress), March 2015.

6.2. Informative References

[OpenID-Connect]
Sakimura, N., Bradley, J., Jones, M., de Madeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014.

Authors' Addresses

Stefanie Gerdes
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Olaf Bergmann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63904
Email: bergmann@tzi.org

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

T. Hardjono
MIT
N. Smith
Intel Corp

July 3, 2015

Fluffy: Simplified Key Exchange for Constrained Environments
draft-hardjono-ace-fluffy-01

Abstract

This document proposes a simplified key exchange protocol for the establishment of a symmetric key shared between two devices or entities within a constrained environment. The pair-wise key establishment is performed using the mediation of a trusted Simple Key Distribution Center (SKDC) entity. The protocol also supports the mediated distribution of a group-key among multiple devices or entities for the purposes of protecting multicast messages.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

This document proposes a simplified key exchange protocol for constrained environments for the establishment of a symmetric key shared between two constrained devices. The pair-wise key establishment is performed using the mediation of a trusted Simple Key Distribution Center (SKDC) entity. The protocol also supports the mediated distribution of a group-key among multiple devices or entities for the purposes of protecting multicast messages.

The simplified key exchange protocol is referred to here as "Fluffy" and is based on a reduced set of Kerberos [RFC4120] messages, adjusting the message flows, types and features to the needs and capabilities of constrained devices and environments. It does not seek to be backward compatible with Kerberos implementations.

The protocol aims to be independent of the underlying transport protocol, and as such the protocol messages are integrity-protected against modifications in-transit. Similar to Kerberos [RFC4120], messages that carry sensitive information (such as keys and/or keying material) are protected using authenticated-encryption. Non-sensitive fields of the messages are integrity-protected using checksums or keyed-hash in the manner of RFC3961. A separate specification will be developed to address in more detail these cryptographic aspects of the current proposed protocol.

Two families of protocol messages are defined here:

- Pairwise key establishment between two entities: When a client seeks to establish a pairwise shared key (called the session encryption key) with a service principal (SP), it invokes the mediation of the SKDC. A four (4) message flow among the client, SKDC and SP are used to establish the pairwise shared key. A further two messages are used to delete the key prior to its expiration.

- Group-shared key establishment among multiple entities: When a client (e.g. client#1) seeks to create a group-shared key (called the group encryption key), it invokes the SKDC to create the group-key, to retain a copy at the SKDC and to return a copy to the requesting client. The distribution of the group-key to other members of a multicast group uses a simple fetch/deliver model in

which new group members (e.g. client#2) must ask for a copy of the group-key from the SKDC.

An additional set of exchanges are introduced to support the delivery of a public key pair to a client entity, with or without an accompanying digital certificate.

The current simplified key exchange protocol does not address the initial secret establishment between an entity and the SKDC. This is referred to in RFC4120 and RFC6113 as "pre-authentication". We anticipate that many types of constrained devices would need to undergo "on-boarding" into an operational state within a constrained environment, and that the on-boarding process may include (directly or as a side-effect) the establishment of the initial secret between the new device and the SKDC already operating in the environment. Thus, for example, the on-boarding process of a device (e.g. door-lock) into a constrained environment (e.g. home basement) with an SKDC entity (e.g. within the alarm panel) may consist of the device and the SKDC running a Diffie-Hellman exchange with the assistance of the human owner. The topic of on-boarding and off-boarding of devices is outside the scope of the current specification.

In this specification we assume that a transport such as CoAP [RFC7252] will be deployed in constrained environments where the IP protocol is operating at the network layer. Environments that are using non-IP transport are out of scope currently for this specification.

The current protocol uses JSON [RFC7159] and CBOR [RFC7049] for its message format. This is in-line with the RESTful paradigm and provides the greatest flexibility for the current protocol to be integrated with other protocols such as OAuth2.0 [RFC6749] for authorization and UMA [UMACORE] for user-centric consent management.

Since the intended deployment environment for the current protocol is a constrained environment, devices and entities there are assumed to use the UUID as the basis for identification. How a device is assigned a UUID is out of scope for the current specification.

The current specification acknowledges that in certain types of constrained environments there is the need for devices to not only operate autonomously for long periods of time, but also for devices to have the capability to take-on different roles with respect to other devices in the environment. Thus, a device (D1) acting as a client to another device (D2) that is acting as an SKDC could also be acting as an SKDC for yet a third device (D3). Thus, the device D1 may have the capability to be both a client and SKDC depending on the operational environment.

As in many deployment environments generally, often security is a trade-off among several factors (e.g. usability, assurance levels, cost, economic risk/benefits, and others). As such, it is realistic to acknowledge that the degree of trustworthiness of an SKDC is dependent on the value of the data and connections within the deployment environment. Thus, an SKDC within a home environment may not be expected to feature the same level of resistance to attacks as an enterprise deployment of a Kerberos KDC.

1.1. Notational Conventions

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this document are to be interpreted as described in [RFC2119].

Unless otherwise noted, all protocol properties and values are case sensitive. JSON [JSON] data structures defined by this specification MAY contain extension properties that are not defined in this specification. Any entity receiving or retrieving a JSON data structure SHOULD ignore extension properties it is unable to understand. Extension names that are unprotected from collisions are outside the scope of this specification.

1.2. Terminology

The current specification seeks to share terminology as much as possible with the terminology defined for CoAP [RFC7252]. However, since the intended Application(s) play a crucial role within constrained networks, we also refer to terminology used by OAuth 2.0 and UMA. Note that within a given constrained network, an device make take multiple roles (client or server) depending on the exchange and layers of the exchange in which they participate.

client
: The client is the entity in a constrained environment seeking to share a key pair-wise with the service principal.

service principal
: The entity with whom the client seeks to establish a pair-wise symmetric key is refereed to as the service principal (SP). This terminology is used to avoid confusion as much as possible with the generic term "server" or "service".

simple key distribution center
: The simple key distribution center (SKDC) is the entity which mediates the establishment of the pair-wise shared key between the client and service principal.

miniticket
: This is the data structure that carries the symmetric key to be shared between the client and service principal.

receipt
: This is the data structure that carries the symmetric key from the SKDC to an entity (client or service principal).

authenticator
: This is the data structure that carries proof-of-possession of a shared symmetric key between two entities.

pair-wise shared key
: A pair-wise shared key (PSK) is symmetric key shared only between a client and service principal.

group shared key
: A group shared key (GSK) is symmetric key shared by two or more entities.

session encryption key
: The session encryption key is the symmetric key generated by the SKDC to be shared pair-wise between the client and the service principal. A session encryption key is an instance of a PSK.

group encryption key
: The group encryption key is the symmetric key generated by the SKDC to be shared among members of a multicast group. A group encryption key is an instance of a GSK.

secret key
: The secret key is the symmetric key that is uniquely shared pair-wise between a client (or service principal) and the SKDC. This term is borrowed from RFC4120. Thus, the client secret key is the symmetric key that is uniquely shared pair-wise between the client and the SKDC. The SP secret key is the symmetric key that is uniquely shared pair-wise between the SP and the SKDC.

set-up keying material
: The cryptographic keying material (including possibly keys) resulting from the initial on-boarding process of a device into a constrained environment is referred to generally as "set-up keying material".

permissions and access control

The permissions and access control (PAC) is the set of information pertaining to permissions for entities within a constrained environment.

resource
The resource refers to the end-point at the service principal to which the application seeks access.

1.3. Design Considerations and Assumptions

There are a number of design considerations and background for the current protocol.

Transport: We assume that the entities in the constrained environment are deploying the CoAP protocol as transport [RFC7252]. However, the design of the current protocol seeks to be transport-independent as much as possible because we anticipate that not all constrained networks may be running CoAP.

JSON data structures: The data structures in this specification are expressed in JSON. We believe this provides the greatest flexibility for the protocol to be integrated into existing protocols for authorization (such as OAuth2.0 [Oauth2] and OpenID-Connect [OIDC]) and consent management by the resource/device owner (such as the User Managed Access (UMA) protocol [UMACORE]).

On-boarding and off-boarding: We assume that constrained devices will undergo the phase of "on-boarding" into a constrained environment. Similarly, "off-boarding" will be required when a constrained device leaves (or is removed) from a constrained environment. The notion of on-boarding is closely related to that of "take-ownership" of certain types of devices. Technologies such as the TPM [TPM] and EPID [EPID] play a crucial role in providing both cryptographic proof (technical trust) and human proof (social trust) in the process of changing the ownership of a device when it is activated and introduced into a constrained environment. We see a close relationship between on-boarding and the current protocol for establishing PSKs and GSKs within a constrained environment.

Secret key establishment or derivation: Following the on-boarding process of a client (resulting in the client and SKDC possessing set-up keying material), the client and the SKDC are assumed to generated the secret key which is shared pair-wise between the client and the SKDC. Methods include using PRFs and other one-way functions. The exact process of generating a secret key from the set-up keying material is out of scope of the current specification. As such, the current Fluffy protocol begins with

the assumption that each entity (client and service principal) already shares pair-wise a secret key with the SDKC. This secret key should be used only for key-management related messages as defined in this specification. Additionally, in this specification we have avoided the use of the term "long-term key" to refer to this secret key due to the broad meaning of this term.

Realms and zones: We have borrowed the notion of "realms" from RFC4120 because we anticipate that a constrained environment may consists of one or more physical networks, and which may be arranged (logically or physically) into "zones". Furthermore, we anticipate that in some use-cases the notion of a "realm" or "zone" may be more ephemeral than is commonly understood for RFC4120 deployments. Thus, there may be constrained use-cases where realms or zones are short-lived.

1.4. Out of Scope and Non-Goals:

The following are out of scope (non-goals) for the current specification:

Authorization and permissions: The issue of permissions and authorization is out of scope for this specification. However, the current specification anticipates the close integration between permissions, authentication and key establishment.

Discovery: Discovery of endpoints, identities, services and other aspects of a constrained environment is out of scope for the current specification.

Backward compatibility with Kerberos: It is not a goal of this specification to achieve backward compatibility with RFC1510 or RFC4120. Similarly, it is not the goal of this specification to be compatible with the MS-PAC [MSPAC] and MS-KILE [MSKILE] specifications.

Pre-authentication: RFC4120, RFC4556 and RFC613 uses the term "pre-authentication" to denote a client obtaining keying material for its secret key prior executing the Kerberos. It is not a goal of this specification to address pre-authentication.

Channel Binding for TLS and DTLS: Channel binding [RFC5929] for DTLS or TLS are out of scope in the current specification.

Certificate Issuance and Management: Issuance of X509 digital certificates and certificate management (in the sense of RFC2459, RFC2797 and RFC4210) is out of scope in the current specification.

2. Common Building Blocks

The current protocol employs a number of data structures that are common across several message types. A number of these data structures have semantic equivalents in RFC4120, while some are newly introduced.

Depending on the message type, some fields may be overloaded in its usage. For example, in the PSK-Request message the client states the identity and realm of the service principal within the SKDC-REQ-BODY. This is similar to RFC4120 because three (3) parties are involved in a PSK establishment (initiated by the client sending a PSK-Request message to the SKDC). However, when the SKDC-REQ-BODY is used in GSK establishment (initiated by an entity sending the SKDC a GSK-Request (GSK-REQ) message) the identity and realm fields are used instead to communicate the desired identity and the realm of the multicast group.

Two building blocks that do not have equivalents in RFC4120 are the Key Data and the Key Envelope structures:

Key Data: The keydata structure is used to convey cryptographic key(s) together with the associated operational parameters for the key(s). The structure of the keydata follows the JSON Web Key (JWK) definition of keys and keying material [RFC7517]. This is a departure from RFC4120.

Key Envelope: The key envelope is used to convey parameters related to a key (e.g. KeyID) but not the key itself. It is used in cases where entities need to refer to a key (e.g. group-key) without having to carry the key in the message.

In the following the common building blocks are discussed.

2.1. SKDC Request Body

The SKDC request body (SKDC-REQ-BODY) carries specific information regarding the type of request and the entities involved in the message. This data structure is used in the initial request send from a client (or SP) to the SKDC.

The SKDC-REQ-BODY varies slightly when used in the PSK-REQ and GSK-REQ messages.

SKDC-REQ-BODY:

- Key Type (kty): This denotes the type of key being requested by the sender (client or SP) of the request.

- Desired algorithm (etype): This is the algorithm that is desired (or supported) by the sender of the request (client or SP) .

- SKDC options - optional (skdc-options): These are flags that are intended for the SKDC only. This field is optional.

- SKDCs realm (skdcrealm): This the name of the realm, domain or zone of the SKDC.

- SKDC's identity (skdcname): This is the identity of the SKDC.

- Service principal's realm - optional (sprealm): This the name of the realm, domain or zone of the service principal in a PSK-REQ message. In a GSK-REQ message it is the realm of the multicast group. This field is optional.

- Service principal's identity (spname): In a PSK-REQ message this is the identity of the service principal. In a GSK-REQ message it is the identity or name of the multicast group.

- Client permissions - optional (cpac): In a PSK-REQ message this is the permisions desired by the client for itself. In a GSK-REQ message this is the permisions desired by the client for the multicast group. This field is optional.

2.2. Miniticket

The miniticket is always created by the SKDC and is always intended for the service principal (although it is delivered via the client who initiates with the PSK-REQ message). The SKDC responds to the client by sending a PSK-Response (PSK-REP) message containing the miniticket and a receipt. The miniticket contains a copy of the session encryption key to be delivered to the service principal by the client in a PSK-Establish (PSK-ESTB) message. As such, the sensitive parts (enc-part) of the miniticket is encrypted using the service principal's secret key (which it shares pair-wise with the KDC). The miniticket is functionally equivalent to the service-ticket in RFC4120.

The miniticket contains the following:

- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this miniticket.

- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this miniticket.

- Service principal's realm - optional (sprealm): This the name of the realm, domain or zone of the service principal for whom this miniticket is destined. This field is optional.

- Service Principal's identity (spname): This is the identity of the service principal for whom this miniticket is destined.

- Encrypted miniticket part (enc-part): This is the encrypted part of the miniticket intended for the service principal. It is encrypted using the secret key shared pair-wise between the SKDC and the service principal. The encrypted part contains the following:

  * Ticket flags - optional (tflags): This is the flags set by the SKDC for the service principal concerning this ticket. This field is optional.

  * Client's realm (crealm): This the name of the realm, domain or zone of the client with whom the receiver of this miniticket shares the enclosed key.

  * Client's identity (cname): This is the identity of the client with whom the receiver of this miniticket shares the enclosed key.

  * Client permissions - optional (cpac): This is the permissions and access control (PAC) structure containing permissions granted to the client associated with the enclosed key. This field is optional.

  * Time of authentication (authtime): This is the time at the SKDC when it created this miniticket in response to a request.

  * Expiration time of key (endtime): The is the expiration time of the key in this miniticket.

  * Service principal permissions - optional (sppac): This is the permissions and access control (PAC) structure granted to the service principal associated with the enclosed key. This field is optional.

  * Transited realms - optional (transited): This is the set of SKDCs and realms that was involved in the issuance of the miniticket for the service principal. This field is used for cross-realm ticket issuance. This field is optional.

  * Key data (keydata): This fields contains the key-data structure that carries the cryptographic key and other parameters

necessary for operating the key. See section Section 2.6 for the key-data structure.

2.3. Receipt

The receipt is always created by the SKDC and is used for the SKDC to deliver a key to a requesting party (be it client, service principal or multicast group members). The receipt is functionally equivalent to the SKDC-Response part in RFC4120.

In The PSK estabslishment flows the receipt is used to deliver the new session encryption key to the requesting client in a PSK-REP message from the SKDC.

In The GSK establishment flows the receipt is used to deliver the new group encryption key to the requesting entity using the GSK-Response (GSK-REP) message. Similarly, members of a multicast group must individually request a copy of the group key using the GSK-Fetch message (GSK-FET), to which the SDKC will send a receipt structure containing a copy of the group key via the GSK-Deliver (GSK-DLVR) message.

When used in a PSK-REP message as a response to the client’s request for a new session encryption key, the receipt names the service principal who is to share the key with the client. The service principal identified in this receipt is the same as that stated in the matching miniticket.

When used in a GSK-REP message for a new group-key creation, the receipt instead names the multicast group with associated with this key. Note that the GSK-REP message has no accompanying miniticket because the SKDC is reponding solely to the requester of the new group-key in a 2-party flow. The receipt in a GSK-Deliver (GSK-DLVR) message (to deliver a copy of the group-key to members of the multicast group) is functionally identical to the receipt in the GSK-REP message.

A note about convention: in the remainder of this specification the entity that requests the creation of a group-key is denoted as the service principal (SP). The members of the multicast group are denoted as the client.

Receipt:

- Receipts flags - optional (rflags): This is the flags set by the SKDC concerning this receipt. This field is optional.

- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this receipt.

- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this receipt.

- Realm - optional (sprealm or mcastrealm): This field is optional, and is used as follows:

  * sprealm: In a PSK-REP message, sprealm is used. This is the name of the realm, domain or zone of the service principal with whom the client (recipient of this receipt) shares the enclosed session encryption key.

  * mcastrealm: In a GSK-REP message and GSK-DLVR message, the mcastrealm is used. This is the realm of the multicast group associated with the enclose group encryption key.

- Name of entity sharing this key (spname or mcastname):

  * spname: In a PSK-REP message, spname is used. This is the identity of the service principal who will be sharing the enclosed session encryption key with requesting client.

  * mcastname: In a GSK-REP message and GSK-DLVR message, mcastname is used. This is the identity of the multicast group associated with the enclosed group encryption key.

- Service Principal's SKDC - optional (spskdc): This field is optional. In a PSK-REP message, this is the identity of the service principal's SKDC. This field is absent in receipts used in a GSK-REP message or GSK-DLVR message.

- Permissions - optional (cpac or grppac): This field is optional. This is the permissions and access control (PAC) structure containing permissions granted, associated with the enclosed key.

  * cpac: In a PSK-REP message, the permissions pertains to the client who requested the session encryption key.

  * grppac: In a GSK-REP message or GSK-DLVR message, the permissions pertain to the members of the multicast group who share this common group encryption key.

- Time of authentication (authtime): This is the time at the SKDC when it created this receipt.

- Nonce from the sender’s request (nonce): This is the nonce found in the previous request message (either PSK-REQ message or GSK-REQ message).

- Expiration time of key (endtime): The is the expiration time of the key in this receipt.

- Key data (keydata): This fields contains the key-data structure that carries the cryptographic key and other parameters necessary for operating the key. In a PSK-REP message, this key is the session encryption key to be shared between the client and service principal. In a GSK-REP message or GSK-DLVR message this is the group encryption key to be shared by the multicast group members. See section Section 2.6 for the key-data structure.

2.4. Authenticator

The authenticator is used by a sender to provide proof-of-possession (POP) to a receiver of a given key that the sender shares with the receiver. The authenticator here is functionally equivalent to the authenticator in RFC4120.

In the PSK-REQ and GSK-REQ messages, the requesting entity uses the authenticator to "authenticate" itself to the SKDC by providing proof-of-possession of the secret key which it shares pair-wise with the SKDC. Note that this mode of usage of the authenticator departs from RFC4120 where a key request message (namely the AP-REQ in RFC4120) is not accompanied by an authenticator.

In the PSK establishment flows, the authenticator is used also in the PSK-ESTB message that is sent from the client to the service principal. More specifically, in the PSK-ESTB message the client encrypts the authenticator using the session encryption key that the client obtained in the previous PSK-REP message it received from the SKDC. Here the authenticator proves to the service principal that the client knows the session encryption key that is enclosed in the accompanying miniticket.

The authenticator is also used in the PSK deletion and GSK deletion flows where the SKDC accompanies its PSK-DELT and GSK-DELT messages respectively with an authenticator that proves to the intended recipient of the message that the SKDC knows the secret key shared pair-wise with that recipient. As such, the authenticator can be used as a generic mechanism to provide proof-of-possession of a shared key between two entities.

Using the example of a client sending an authenticator to a service principal, the authenticator contains the following:

- Client’s realm (crealm): This the identity of the realm, domain or zone of the client that created the authenticator.
- Client’s identity (cname): This is the identity of the client that created the authenticator.
- Client’s current time (ctime): This is the time at the client when it created the authenticator.
- Nonce (nonce): This is a new nonce generated by the client (for the recipient of the authenticator).
- Sequence number - optional (seqnum): This is the sequence number used by the client to detect attacks. This field is optional.
- Checksum - optional (cksum): This is the keyed-checksum (based on the session encryption key) used by the client as sender. This field is optional.

2.5. Acknowledgement

The acknowledgement structure (ack) is used by one entity to send a positive acknowledgement to another entity regarding a message that was previously exchanged. The acknowledgement would carry a nonce that was found in the related previous message. The type of acknowledgement is expressed in the enveloping header message (e.g. PSK-ACK type acknowleges a previous PSK-ESTB message).

Note that the acknowledgement structure is intended to be generic, and as such can also be used by the SKDC to send an acknowledgement to a client or service principal.

Using the example of a service principal (sender) sending the an acknowledgement to a client (receiver), the acknowledgement structure contains the following:

- Service principal’s realm - optional (sprealm): This the identity of the realm, domain or zone of the service principal who created this acknowledment. This field is optional.
- Service Principal’s identity (spname): This is the identity of the service principal for who created this acknowledgement.
- Nonce from the client’s previous message (nonce): This is the nonce found in the previous message being acknowledged.
- Time of authentication (authtime): This is the time at the service principal when it created this acknowledgement message.

- Sequence number - optional (seqnum): This is the sequence number used to detect attacks. This field is optional.

2.6. Key Data

The key-data structure is used to carry cryptographic keys and the related parameters needed to operate the keys. The construction of the key-data structure follows that of the JSON Web Keys [RFC7517] which supports not only symmetric keys, but also public key pairs and X509 certificates.

The intent is to support the delivery of either a single symmetric key, a public key pair or one key (half) of a public key pair.

Delivery of an array of keys is for future consideration.

2.6.1. Symmetric Key Data

The key-data structure for carrying a symmetric key is as follows.

- Key type (kty): This is the type of key contained in the current key-data structure.

- Key ID - optional (keyid): This is the name or handle for the key that can be be referred to by parties sharing they key. This field is optional.

- Key (key): This is the symmetric key.

- Key operations parameters (keyops): This is parameters required to operate the cryptographic key.

- Algorithm (alg): This is the algorithm name for the use of the key.

2.6.2. Asymmetric Key Data

The key-data structure for carrying an asymmetric key and parameters is as follows.

- Key type (kty): This is the type of key contained in the current key-data structure.

- Key ID - optional (keyid): This is the name or handle for the key that can be be referred to by parties sharing they key. This field is optional.

- Key (key): This is the public key.

- Key operations parameters (keyops): This is parameters required to operate the cryptographic key.
- Algorithm (alg): This is the algorithm name for the use of the key.
- Public key usage (pkuse): For public keys this field indicates the use of the key. See RFC7517.
- X509 URL parameter (x5u): This parameter is a URI [RFC3986] that refers to a resource for an X.509 public key certificate or certificate chain [RFC5280]. See RFC7517.
- X509 certificate chains parameter - optional (x5c): This parameter contains a chain of one or more PKIX certificates [RFC5280]. See RFC7517.
- X509 thumbprint parameter - optional (x5t): This parameter contains a base64url-encoded SHA-1 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. See RFC7517.
- X509 SHA256 thumbprint parameter - optional (x5t256): This parameter contains a base64url-encoded SHA-256 thumbprint (a.k.a. digest) of the DER encoding of an X.509 certificate [RFC5280]. See RFC7517.

2.7. Key Envelope

The key envelope structure is used in messages that refer to a cryptographic key by its KeyID. As such, the key envelope structure by definition must never carry any cryptographic keys or keying material.

The key envelope is used primarily in the deletion of a a PSK or a GSK. When the SKDC wishes to delete a given PSK that it shares with an entity, it can refer to the target key by way of the KeyID in the key envelope.

In order to delete a PSK that a client and service principal shares (through the mediation of the SKDC via a previous 3-party PSK establishment flow), the SKDC must separately delete the PSK at the client and at the service principal (i.e. using two separate PSK-Delete (PSK-DELT) messages).

The key envelope is encrypted using the secret key shared between the SKDC and the entity (client or service principal) for whom the envelope is destined.

The key envelope must never be encrypted using the target key that is to be deleted.

The key envelope structure contains the following:

- Envelope Flags - optional (envflags): This is the flags related to the envelope. This field is optional.
- Issuing SKDC's realm (skdcrealm): This the name of the realm, domain or zone of the SKDC that issued this envelope.
- Issuing SKDC's identity (skdcname): This is the identity of the SKDC that issued this envelope.
- Current time (authtime): This is the time at the SKDC when it created the encrypted envelope.
- Nonce (nonce): This is a new nonce generated by the SKDC (for the recipient of the envelope).
- Sequence number - optional (seqnum): This is the sequence number to detect attacks. This field is optional.
- Key data (keydata): This is the key-data structure which contains the KeyID of the target key. The key-data in a key envelope must not contain any cryptographic keys. See section Section 2.6 for the key-data structure.

3. Pair-wise Shared Key Establishment

This section describes the pair-wise key establishment between the client and the service principal.

3.1. Basic Protocol Exchange

Prior to executing the Fluffy protocol, a client must first be in possession of a secret key that it shares pair-wise with the SKDC. The process or method of obtaining the client secret key is outside the scope of the current specification, but for a device operating within a constrained environment this may be a direct consequence of the on-boarding or take-ownership process.

The PSK establishment consists of a 4-message flow from the client to SKDC, the SKDC back to the client, and between the client and the service principal.

Note that unlike RFC4120, the client must provide an authenticator in its first message (PSK-Request) to the SKDC. This authenticator is

encrypted by the client using the secret key it shares pair-wise with the SKDC.

The message flows consists of the following steps and are summarized in Figure 1.

- PSK-Request (PSK-REQ): The client sends a PSK-Request message to the SKDC asking for the SKDC to mediate the sharing of a new session encryption key between the client and the service principal. The client must indicate the intended service principal in this message. Unlike RFC4120 the client must include an authenticator that is encrypted to the SKDC as a proof of possession of the secret key it shares with the SKDC.

- PSK-Response (PSK-REP): The SKDC responds by generating a new session encryption key and placing the key into a miniticket intended for the service principal. The miniticket is encrypted using the secret key which is pair-wise shared only between the SKDC and the service principal. Additionally, the SKDC places a copy of this new session encryption key into a receipt structure, and encrypting it using the secret key pair-wise shared between the SKDC and the client. Both the miniticket and the receipt are then returned to the client. The client must forward the miniticket unmodified to the service principal in the PSK-Establish message.

- PSK-Establish (PSK-ESTB): The client then decrypts the receipt to obtain the session encryption key. The client uses this session encryption key to encrypt an authenticator structure as proof of possession for the service principal. The client then sends the authenticator and the miniticket (unmodified from the SKDC) to the service principal. The service principal decrypts the miniticket to obtain the session encryption key, and then it uses the session encryption key to verify the authenticator (by decrypting it). At this point the client and the service principal shares the session encryption key.

- PSK-Acknowledge (PSK-ACK): The service principal exercises the newly received session encryption key by encrypting an acknowledgement message to the client.

Similar to RC4120, the integrity of the messages containing cleartext data is protected using a checksum mechanism (e.g. keyed hash) based on the client’s secret key [RFC3961].

The PSK Establishment Flows

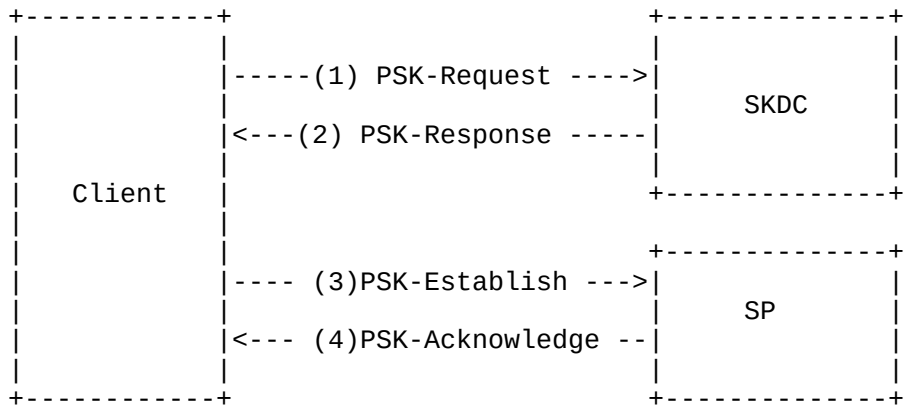


Figure 1

The message components as used in the protocol are summarized in Figure 2. Note that all protocol messages are integrity-protected, and some are encrypted.

The PSK Message Components

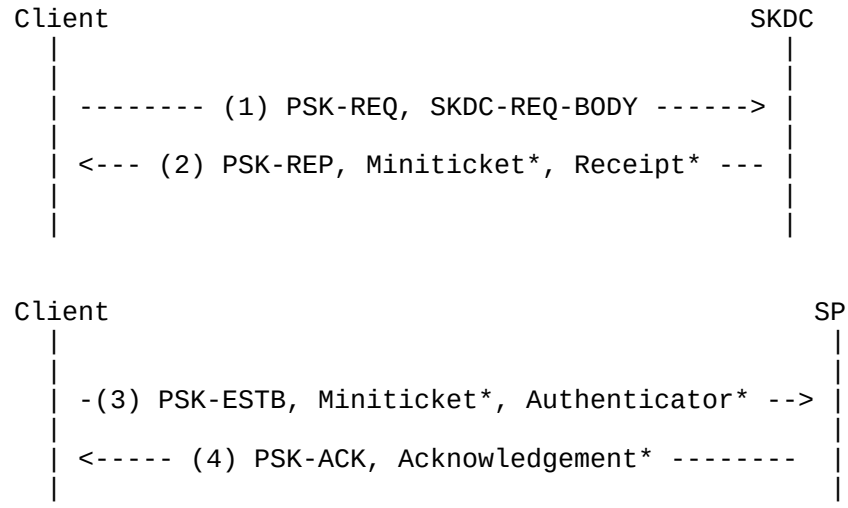


Figure 2

3.2. PSK-Request Message (PSK-REQ)

The PSK-Request (PSK-REQ) message is sent from the client to the SKDC asking for the SKDC to mediate the establishment of a pair-wise shared key between the client and the service principal. The client must indicate the intended service principal in this message.

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-REQ".
- SKDC request body (req-body): The request body contains the parameters required by the SKDC to mediate key establishment for the client. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a PSK-REQ message contains the following:
  - Key Type (kty): This is type of key being requested by the client from the SKDC.
  - Desired algorithm (etype): This is the algorithm that is desired (or supported) by the client.
  - SKDC options - optional (skdc-options).
  - SKDC’s realm (skdcrealm).
  - SKDC’s identity (skdcname).
  - Service principal’s realm - optional (sprealm).
  - Service principal’s identity (spname).
  - Client permissions - optional (cpac).
- Authenticator (authenticator): This is the authenticator encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information on the authenticator structure.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.3. PSK-Response Message (PSK-REP)

The PSK-Response (PSK-REP) is sent by the SKDC to the client as a response to the client's previous PSK-REQ message. The PSK-REP message carries two crucial data structures, namely the miniticket and the receipt.

The miniticket here is intended solely for the service principal and carries a copy of the session encryption key (key) intended for the service principal. As such the miniticket is encrypted to the service principal using the secret key shared between the SKDC and service principal. Although the miniticket is returned to the client, the client is unable to view or modify the encrypted parts of the miniticket.

The receipt here is intended solely for the client and carries a copy of the session encryption key (key) intended for the client. The receipt is encrypted to the client using the secret key shared between the SKDC and client.

The PSK-REP message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-REP".
- Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.
- Client's identity (cname): This is the identity of the client.
- Miniticket (mticket): The miniticket contains the following:
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Service principal's realm - optional (sprealm).
  - Service Principal's identity (spname).
  - Encrypted miniticket part (enc-part): This is the part of the PSK-REP message that is encrypted by the SKDC to the service principal, using the secret key that the SKDC shares pair-wise with the service principal. It contains the following:
    - Ticket flags - optional (tflags).

+ Client’s realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.

+ Client’s identity (cname).

+ Client permissions - optional (cpac).

+ Time of authentication (authtime).

+ Expiration time of this key (endtime).

+ Service principal permissions - optional (sppac).

+ Transited realms - optional (transited).

+ Key data (keydata): This key-data structure contains a copy of the session encryption key destined for the service principal. See section Section 2.6 for the key-data structure.

o Receipt (receipt): This is the part of the PSK-REP message that is encrypted by the SKDC to the client, using the secret key that the SKDC shares pair-wise with the client. It contains the following:

* Receipts flags - optional (tflags).

* Issuing SKDC’s realm (skdcrealm).

* Issuing SKDC’s identity (skdcname).

* Service principal’s realm - optional (sprealm).

* Service Principal’s identity (spname).

* Service Principal’s SKDC - optional (spskdc).

* Client permissions - optional (cpac).

* Time of authentication (authtime).

* Nonce from the Client’s previous PSK-REQ request message (nonce).

* Expiration time of this key (endtime).

    * Key data (keydata): This key-data structure contains a copy of the session encryption key destined for the client. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.4. PSK-Establish Message (PSK-ESTB)

The PSK-Establish (PSK-ESTB) message is sent from the client to the service principal requesting it to share a key (i.e. the session encryption key). The PSK-ESTB message contains two parts. The first is the miniticket obtained by the client from the SKDC in the previous PSK-Response (PSK-REP) message.

The second is the authenticator created by the client. The authenticator is encrypted using the session encryption key which the client obtained in the receipt from the previous PSK-REP from the SKDC).

The authenticator proves to the service principal that the client is in possession of the session encryption key.

This PSK-ESTB message is sent by the client to the service principal. It contains the following:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "PSK-ESTB".

- Client’s realm (crealm): This the name of the realm, domain or zone of the client.

- Client’s identity (cname): This is the identity of the client.

- Miniticket (mticket): This is the miniticket structure (unmodified) that the client received from the SKDC in the previous PSK-REP message. See Section Section 3.3 for the miniticket in the PSK-REP message.

- Authenticator: The authenticator in the PSK-ESTB contains the following:

    * Client’s realm (crealm).

    * Client’s identity (cname).

    * Client’s current time (ctime).

    * A new nonce generated by the client for the service principal (nonce).

    * Sequence number - optional (seqnum).

    * Checksum - optional (cksum).

  o Extensions - optional (ext): Reserved for future extensions. This field is optional.

3.5. PSK-Acknowledge Message (PSK-ACK)

The PSK-Acknowledge (PSK-ACK) message is sent from the service principal to the client in response to the previous PSK-ESTB message.

The message contains an acknowledgement part that is encrypted by the service principal using the session encryption key (which the service principal obtained in the miniticket in the previous PSK-ESTB message).

The PSK-ACK message contains the following:

  o Protocol version (pvno): This the version of the protocol.

  o Message type (msg-type): The message type for this message is "PSK-ACK".

  o Client’s realm (crealm).

  o Client’s identity (cname).

  o Acknowledgement (ack): This is the acknowledgement structure that contains the following:

    * Service principal’s identity (spname).

    * Service principal’s realm - optional (sprealm).

    * Nonce from the Client’s previous PSK-ESTB message (nonce).

    * Time of authentication (authtime).

    * Sequence number (inremented) from PSK-ESTB message - optional (seqnum).

- o Extensions - optional (ext): Reserved for future extensions. This field is optional.

4. Pair-wise Shared Key Deletion

The current protocol supports the proactive deletion by the SKDC of a pairwise shared key (PSK) prior to the expiration of the key. The target PSK to be deleted must be a PSK that a client and service principal had established through the mediation of the SKDC using the PSK establishment flow.

Only the SKDC has the authority to send a key-deletion message (PSK-DELT) to an entity (client or service principal). A client or service principal MUST ignore key-deletion messages which did not come from the SKDC.

The SKDC authenticates itself to the client (service principal) by including a key envelope that is encrypted using the secret key shared between the SKDC and the client (service principal). The key envelope identifies the target key to be deleted by its key-ID.

If a PSK-DELT message issued by the SKDC is received at the client after the named key has in fact expired, the client must still respond with a PSK-Delete-Confirm (PSK-DELC) message. This confirms to the SKDC that the named key no longer exists at the client.

In order to delete a PSK that a client and service principal shares (through the mediation of the SKDC via a PSK establishment flow), the SKDC must delete the PSK at the client and at the service principal separately (using two separate PSK-DELT messages respectively).

The message components as used in the protocol are summarized in Figure 3.

The PSK Delete Message Components

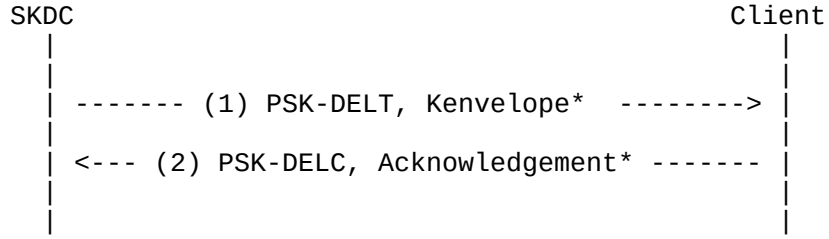


Figure 3

4.1. PSK-Delete Message (PSK-DELT)

The PSK-DELT message is sent from the SKDC to a client (or service principal) asking for the delition or erasure of the PSK identified in the message. The SKDC must indicate the intended recipient in this message.

- Protocol version (pvno): This is the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-DELT".
- Client's realm (crealm): This the identity of the realm, domain or group in which the client belongs in connection to this request.
- Client's identity (cname): This is the identity of the client.
- Key Envelope (kenvelope): The key envelope is encrypted using the client's secret key that it shared with the SKDC. The key envelope contains the following:
  - Envelope Flags - optional (envflags).
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Current time (authtime).
  - New nonce generated by the SKDC (nonce).
  - Sequence number - optional (seqnum).
  - Key data (keydata): This key-data structure contains the KeyID of the target key to be deleted. The key-data in a key envelope must never contain a cryptographic key. See section Section 2.6 for the key-data structure.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

4.2. PSK-Delete-Confirm Message (PSK-DELC)

The psk-delete-confirm (PSK-DELC) message is sent from the client (or service) to the SKDC confirming the removal of the PSK identified in the previous PSK-DELT message. A client (or service principal) must only send the PSK-DELC message after it has successfully remove or erase the target key from its key store.

The acknowledgement in the PSK-DELC message is encrypted by the client using the secret key that the client shares with the SKDC. See Section Section 2.5 for more information on the acknowledgement structure.

The PSK-DELC message contains the following:

- Protocol version (pvno): This is the version of the protocol.
- Message type (msg-type): The message type for this message is "PSK-DELC".
- SKDC's realm - optional (skdcrealm).
- SKDC's identity (skdcname).
- Acknowledgement (ack):
  - Client's identity (cname).
  - Client's realm - optional (crealm).
  - Nonce from the SKDC's previous PSK-DELT message (nonce).
  - Client's current time (authtime).
  - Sequence number - optional (seqnum): This field is optional.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5. Group Shared Key Establishment

The current protocol supports the establishment of a group-shared symmetric key (referred to as the "group encryption key" or "group-key") among a number of entities within a constrained environment. The group encryption key affords group-authenticity for messages but not source-authenticity since the symmetric key is shared among multiple entities (members of the multicast group). See RFC3740 for further discussion regarding multicast group security.

In the following we use the notation of the service principal (SP) as the entity that initiates the multicast group creation by requesting the SKDC to create a new group encryption key and to maintain a copy of that group-key until such time it expires or is deleted. The clients that request and obtain a copy of the group-key are denoted as "members" of the group. The current specification follows the

convention that only the group-creator and the SKDC are permitted to proactively delete a group encryption key.

Using the service principal as the group-creator, the service principal must accompany its request to the SKDC with an authenticator.

Each group encryption key is associated with an owner (creator) who requested its creation at the SKDC. When an SP seeks to establish a new group encryption key, it sends a GSK-Request message to the SKDC asking that the SKDC generate a new symmetric key (i.e. the group encryption key), return a copy of the group encryption key to the service principal (via a receipt inside a GSK-Response message) and for the SKDC to retain a copy of the group key (for subsequent fetches by the clients). The sensitive parameters of the GSK-Response message (including the group encryption key) inside the receipt is encrypted using the secret key pair-wise shared between the SP and the SKDC.

When a client seeks to obtain a copy of a group encryption key associated with a multicast group, the client sends a GSK-Fetch message to the SKDC identifying the multicast group (mcastname) of interest. The requesting client must accompany the request with an authenticator that is encrypted using the secret key shared between the client and the SKDC.

If a corresponding group or group encryption key does not exist at the SKDC, the SKDC returns an error message. Otherwise, the SKDC returns a copy of the group encryption key (inside a receipt) to the requesting client using the GSK-Deliver message. The sensitive parameters of the GSK-Deliver message (including the group encryption key) inside the receipt is encrypted using the secret key pair-wise shared between the requesting client and the SKDC.

The GSK Establishment Flows

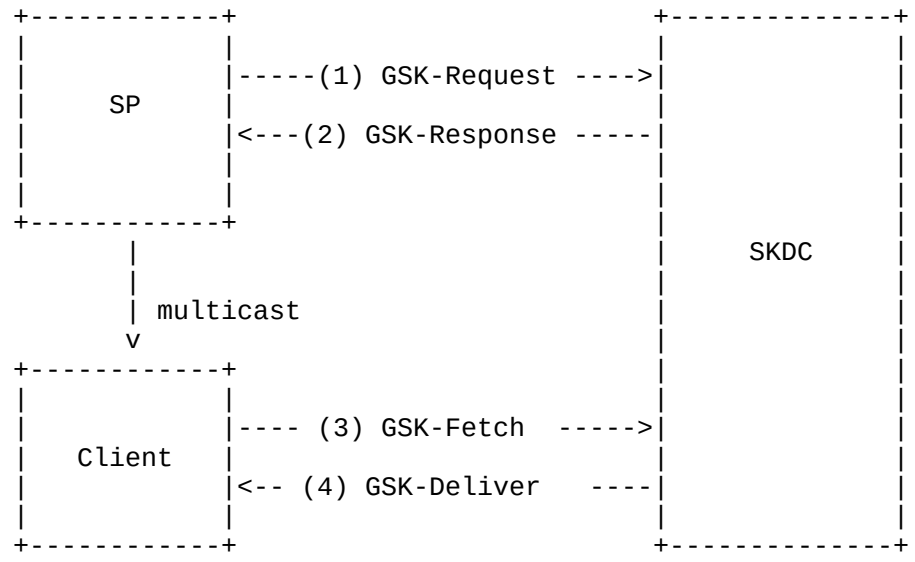


Figure 4

The GSK establishment in the protocol consists of two sets of 2-messages each:

- Creation of the group-key at the SKDC:
  - GSK-Request: A service principal sends a GSK-Request (GSK-REQ) message to the SKDC asking for a new group key to be created. The service principal provides the desired name (mcastname) of the multicast group. It must accompany the request with an authenticator to the SKDC. After generating the new group-key the SKDC retains a copy of the group-key (until it expires) and associates it with the multicast group name (mcastname).
  - GSK-Response: The requesting service principal obtains a copy of the new group-key via the GSK-Response (GSK-REP) message from the SKDC. A receipt structure to carries the group-key. The receipt is encrypted by the SKDC using the secret key it shares with the service principal.
- Fetching of a copy of the group-key from the SKDC:
  - GSK-Fetch: To request a copy of the group-key, a client sends the GSK-Fetch (GSK-FET) message to the SKDC with an

authenticator. The client must indicate the desired multicast group (mcastname) in the GSK-FET message.

* GSK-Deliver: The SKDC returns a copy of the group-key to the client via the GSK-Deliver (GSK-DLVR) message. A receipt structure to carries the group-key. The receipt is encrypted by the SKDC using the secret key it shares with the client.

The message components as used in the protocol are summarized in Figure 5. Note that all protocol messages are integrity-protected, and some are encrypted.

The GSK Message Components

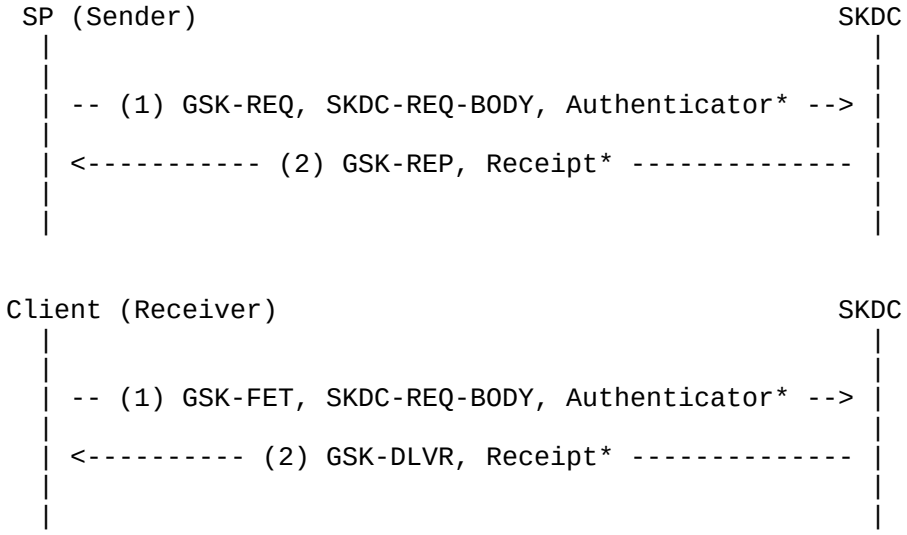


Figure 5

5.1. GSK-Request Message (GSK-REQ)

The GSK-Request message is sent from the service principal to the SKDC asking the SKDC to create a new group-key. The service principal authenticates itself to the SKDC by including an authenticator in the GSK-REQ message.

The contents of the GSK-REQ message is as follows:

o Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-REQ".

- SKDC request body (req-body): The request body contains the parameters related to the group-key and multicast group. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body in a GSK-REQ message contains the following:

  * Key Type (kty): This is type of key being requested. For a group shared key the type is "SYMM".

  * Desired algorithm (etype): This is the algorithm that is desired (or supported) by the service principal.

  * SKDC options - optional (skdc-options).

  * SKDC’s realm (skdcrealm).

  * SKDC’s identity (skdcname).

  * Multicast group realm - optional (mcastrealm): This is the desired realm name associated with the multicast group.

  * Multicast group identity (mcastname): This is the desired identity or name for the multicast group.

  * Group permissions - optional (grppac): This is the desired set of permissions associated with the multicast group.

- Authenticator: In the case of a GSK-REQ message, the authenticator is encrypted by the service principal (SP) using the secret key it shares with the SKDC. The authenticator in the GSK-REQ contains the following:

  * SP’s realm (sprealm).

  * SP’s identity (spname).

  * SP’s current time (sptime).

  * A new nonce generated by the SP (nonce).

  * Sequence number - optional (seqnum).

  * Checksum - optional (cksum).

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.2. GSK-Response Message (GSK-REP)

The GSK-Response (GSK-REP) message is sent from the SKDC to the service principal in response to the service principal's GSK-Request message. The GSK-Response message contains a receipt structure which carries the new group-key for the requesting service principal.

Note that the GSK-REP message does not contain a miniticket.

The Receipt in the GSK-Response message is encrypted by the SKDC to the requesting service principal using the secret key that is shared between the SKDC and the service principal.

The GSK-Response message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "GSK-REP".
- SP's realm (sprealm): This the name of the realm, domain or zone in which the SP belongs in connection to this request.
- SP's identity (spname): This is the identity of the SP requesting the group-key in the previous GSK-REQ message.
- Receipt (receipt): The receipt carries the group-key and relevant parameters. It is encrypted by the SKDC to the SP using the secret key shared between the SKDC and the cSP. The receipt (receipt) contains the following:
  - Receipts flags - optional (rflags).
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Multicast group realm - optional (mcastrealm).
  - Multicast group identity (mcastname).
  - Group permissions - optional (grppac).
  - Current time (authtime).
  - Nonce from the GSK-REQ request (nonce).
  - Expiration time of key (endtime).

    * Group key (keydata): The key-data structure contains the group-key destined for the requesting SP. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.3. GSK-Fetch Message (GSK-FET)

The GSK-Fetch message is sent by a client to the SKDC asking for a copy of a group-key asociated with a multicast group. The client must identify the desired multicast group name (mcastname) in the SKDC-REQ-BODY of the message. The client authenticates itself to the SKDC by including an authenticator.

The contents of the GSK-Fetch message is as follows:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-FET".

- SKDC request body (req-body): The req-body contains the parameters required by the SKDC identify the multicast group. See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a GSK-FET message contains the following:

    * SKDC options - optional (skdc-options).

    * SKDC’s realm - optional (skdcrealm).

    * SKDC’s identity (skdcname).

    * Multicast group realm - optional (mcastrealm).

    * Multicast group identity (spname): This is the name of the multicast group whose group-key is being fetched.

    * KeyID - optional (keyid).

- Authenticator (authenticator): The authenticator in the GSK-FET is encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information the the authenticator structure. The authenticator in the GSK-FET contains the following:

    * Client’s realm (crealm).

    * Client’s identity (cname).

    * Client’s current time (ctime).

    * A new nonce generated by the client (nonce).

    * Sequence number - optional (seqnum).

    * Checksum - optional (cksum).

  o Extensions - optional (ext): Reserved for future extensions. This field is optional.

5.4. GSK-Deliver Message (GSK-DLVR)

The GSK-Deliver (GSK-DLVR) message is sent from the SKDC to the client in response to the client’s GSK-Fetch message. The GSK-Deliver message uses the receipt structure to carry the group encryption key. The receipt is encrypted using secret key which is shared pair-wise between the client and the SKDC.

The contents of the GSK-Deliver message is as follows:

  o Protocol version (pvno): This is the version of the protocol.

  o Message type (msg-type): The message type for this message is "GSK-DLVR".

  o Client’s realm (crealm): This the name of the realm, domain or group in which the client belongs in connection to this request.

  o Client’s identity (cname): This is the identity of the client.

  o Receipt (receipt): This is the part of the GSK-Deliver message carries the group-key. It is encrypted by the SKDC to the client using the secret key shared between the SKDC and the client. The receipt contains the following:

    * Receipts flags - optional (rflags).

    * Issuing SKDC’s realm (skdcrealm).

    * Issuing SKDC’s identity (skdcname).

    * Multicast group realm - optional (mcastrealm).

    * Multicast group identity (mcastname): This is the name of the multicast group whose group-key is being delivered.

  * Group permissions - optional (grppac).

  * Current time (authtime).

  * Nonce from the previous GSK-FET message (nonce).

  * Expiration time of key (endtime).

  * Group key (keydata): This key-data structure contains the group key destined for the requesting client. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

6. Group Shared Key Deletion

The current protocol supports the proactive removal of a group-key associated with a multicast group prior to the expiration of the group-key. Only the creator (owner) of the multicast group can request the SKDC to delete a group-key (or the SKDC itself could perform a group-key deletion in response to an external authorized trigger).

Due to the nature of a symmetric group-key, the removal of a group-key from a multicast group requires the SKDC to issue unicast PSK-Delete messages to each known member of the group. When the SKDC sends the PSK-Delete message to a client who is a group member, the SKDC must identify the group-key via its KeyID. Each group member must respond to the SKDC with a PSK-Delete-Confirm (PSK-DELC) message to confirm the deletion or erasure of the group-key. See Section Section 4.1 for more information on the PSK-DELT and PSK-DELC messages.

The SKDC must wait for all known group members to individually confirm (via a PSK-DELC message) their deletion of the group-key. Only then should the SKDC return a GSK-Delete-Confirmation (GSK-DELC) message to the service principal who requested the group-key deletion.

When the SKDC is in the process of deleting a group-key, it must deny further requests for the group-key from new members.

The group-key deletion process involves four types of messages:

- GSK-Delete (GSK-DELT): The SP (as group-owner) sends a GSK-Delete request to the SKDC.

- PSK-Delete (KeyID): For each member of the group (i.e. clients who previosuly requested a copy of the group-key), the SKDC sends a unicast PSK-Delete (PSK-DELT) message to that client identifying the target key to be deleted (by KeyID). See Section Section 4.1 for the PSK-DELT message.

- PSK-Delete-Confirm (KeyID): Each group member responds after key-deletion with a PSK-DELC message to the SKDC. See Section Section 4.2for the PSK-DELC message.

- GSK-Delete-Confirmed (GSK-DELC): The SKDC responds with a GSK-Delete-Confirmed message to the SP (as group-owner) once all copies of the group-key has been deleted at the group-members.

The message flow for GSK deletion is shown in Figure 6.

The GSK Deletion Message Components

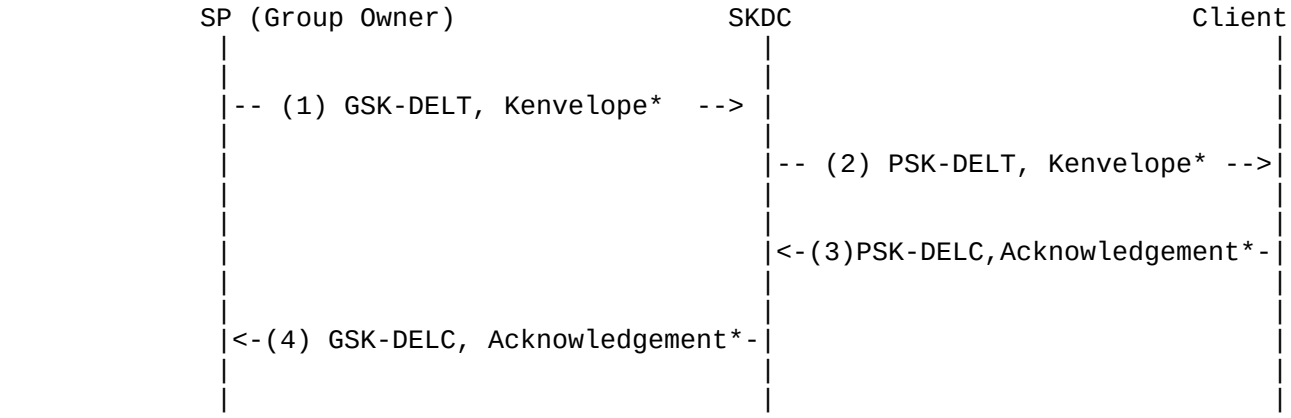


Figure 6

6.1. GSK-Delete Message (GSK-DELT)

The GSK-Delete (GSK-DELT) request message is sent from an SP (group owner) to the SKDC asking for the deletion or erasure of the group-key of the multicst group identified in the message.

The GSK-DELT message contains the following.

- Protocol version (pvno): This is the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-DELT".

- SP's realm (sprealm): This the name of the realm, domain or group in which the SP belongs in connection to this request.

- SP's identity (spname): This is the identity of the SP.

- Key Envelope (kenvelope): The key envelope is encrypted using the SP's secret key that it shares with the SKDC. The key envelope contains the following:

  * Envelope Flags - optional (envflags).

  * Multicast group realm - optional (mcastcrealm): This is the realm of the multicast group whose group-key is being deleted.

  * Multicast group identity (skdcname): This is the name of the multicast group whose group-key is being deleted.

  * Current time (authtime).

  * New nonce generated by the SP (nonce).

  * Sequence number - optional (seqnum).

  * Key data (keydata): This key-data structure contains the KeyID of the target key to be deleted. The key-data in a key envelope must never contain a cryptographic key. See section Section 2.6 for the key-data structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

6.2. GSK-Delete-Confirm Message (GSK-DELC)

In response to a GSK-Delete request from a service principal (as group owner), the SKDC sends a GSK-Delete-Confirm (GSK-DELC) message to the service principal for the successful deletion of not only its copy of the group-key, but also the successful deletion of the copies of the group-key at each group member (client).

The GSK-DELC message contains the following:

- Protocol version (pvno): This is the version of the protocol.

- Message type (msg-type): The message type for this message is "GSK-DELC".

- SP's realm - optional (sprealm).
- SP's identity (spname).
- Acknowledgement (ack): The acknowledgement is encrypted by the SKDC to the SP using the secret key shared only between the SKDC and SP.
  - Multicast group identity (mcastname): This is the name of the multicast group whose group-key has been deleted.
  - Multicast group realm - optional (mcastrealm).
  - Nonce from the SP's previous GSK-DELT message (nonce).
  - Current time (authtime).
  - Sequence number - optional (seqnum): This field is optional.
- Extensions - optional (ext): Reserved for future extensions. This field is optional.

7. Public Key Pair Establishment

The current protocol supports the distribution of public key pairs and certificates. Since the SKDC is not a certificate authority (in the sense of RFC2459), issuing (signing) X509 certificates is out of scope for the current specification. If the SKDC receives from a client a request for a new certificate, the SKDC must obtain a certificate from a CA server (or CA service) located either in the same realm/zone or elsewhere on behalf of the requesting client. How the SKDC discovers CA services is out of scope for the current specification.

In the following we provide additional message types to support a client requesting the SKDC to deliver a new public key pair and to return the key pair to the client.

When a client seeks to obtain a new public key pair, the client sends a Public Key Pair Request (PKP-REQ) message to the SKDC. The requesting client must include an authenticator that is encrypted using the secret key it shares with the SKDC. The SKDC returns the key-pair in the receipt structure to the client (encrypted to the client).

As a further option to the PKP-REQ message, if the client identifies a service principal in the SKDC-REQ-BODY of the PKP-REQ message, the SKDC would also return a miniticket that contains only the public

half of the key-pair. The miniticket would be encrypted by the SKDC to the named service principal, using the secret key that the SKDC shares with the service principal. Note that this is a departure from the PSK-Request semantics (for symmetric key requests) because the miniticket contains a public key (instead of a symmetric key).

The client (or the SKDC) then sends the miniticket to the service principal who can decrypt the miniticket using the secret key it shares with the SKDC.

By encrypting the miniticket to the service principal, the SKDC is effectively attesting to the binding between the public key pair and the client's identity (without the use of digital certificates). The service principal trusts SKDC to provide this pseudo-attestation regarding the client as the owner of the new public key pair.

Note that in this approach the security of the public key pair is only as a secure as the symmetric key algorithm used to encrypt the receipt and the miniticket.

If additionally the client seeks to obtain a digital certificate for the new public key, then the client must indicate this option in the SKDC-Options field (in the SKDC-REQ-BODY) of the PKP-Request message. There are several options available related to digital certificates and CA certificates (trust anchors):

No certificate: This is the default setting for the PKP-Request message.

Certificate: This means the client additionally requests the return of a new X509 certificate corresponding to the new public key.

Include certificate chain: This option is meaningful only of the client requests a new X509 certificate. When set, this option means that the client also requests the certificate chain consisting of copies of all signing certificates to the top of the certificate hierarchy.

7.1. Public Key Pair Request (PKP-REQ)

A client uses the PKP-Request message (PKP-REQ) to request a new public key pair from the SKDC. The client must include an authenticator when sending this message to the SKDC. The PKP-REQ message contains the following:

- Protocol version (pvno): This the version of the protocol.

- Message type (msg-type): The message type for this message is "PKP-REQ".

- SKDC request body (req-body): The request body contains the parameters required by the SKDC in the context of this request. the See Section Section 2.1 for more information on the SKDC request body. The SKDC request body of a PKP-REQ message contains the following:

  * Key Type (kty): This is type of key being requested by the client from the SKDC.

  * Desired algorithm (etype): This is the algorithm that is desired (or supported) by the client.

  * SKDC options - optional (skdc-options): The client sets the "certificate request" option (and possibly the "certificate chain" option) in this field if it requests a digital certificate (and corresponding chain) to be returned together with the public key pair.

  * SKDC’s realm (skdcrealm).

  * SKDC’s identity (skdcname).

  * Service principal’s realm - optional (sprealm):

  * Service principal’s identity -- optional (spname):

    + spname is present: If the spname is present, this means that the client wishes for the SKDC to prepare copy of the new public key only for the named service principal via the miniticket structure.

    + spname is absent: If the spname is absent the SKDC delivers the public key pair only to the client via the receipt structure.

  * Client permissions - optional (cpac).

- Authenticator (authenticator): This is the authenticator encrypted by the client to the SKDC using the secret key that the client shares with the SKDC. See Section Section 2.4 for more information on the authenticator structure.

- Extensions - optional (ext): Reserved for future extensions. This field is optional.

7.2. Public Key Pair Response (PKP-REP)

In response to the PKP-Request message (PKP-REQ) from a client entity, the SKDC returns a copy of the new public key pair to the client in a receipt structure, encrypted using the secret key shared between the client and SKDC. This ensures that only the client is in possession of the new public key pair (notably the private key).

If the SKDC-REQ-BODY of the client's previous PKP-request message contains the identity of a service principal (spname), the SKDC must also create a miniticket containing a copy of the public key only. The miniticket is encrypted to the service principal.

The PKP-REP message contains the following:

- Protocol version (pvno): This the version of the protocol.
- Message type (msg-type): The message type for this message is "PKP-REP".
- Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.
- Client's identity (cname): This is the identity of the client.
- Miniticket - optional (mticket): If the client identified a service principal in the previous PKP-REQ message (in its SKDC-REG-BODY), then a miniticket is included by the SKDC in this PKP-REP message. If present, the miniticket contains the following:
  - Issuing SKDC's realm (skdcrealm).
  - Issuing SKDC's identity (skdcname).
  - Service principal's realm - optional (sprealm).
  - Service Principal's identity (spname).
  - Encrypted miniticket part (enc-part): This is the part of the PKP-REP message that is encrypted by the SKDC to the service principal, using the secret key that the SKDC shares pair-wise with the service principal. It contains the following:
    - Ticket flags - optional (tflags).
    - Client's realm (crealm): This the name of the realm, domain or zone in which the client belongs in connection to this request.

+ Client’s identity (cname).

+ Client permissions - optional (cpac).

+ Time of authentication (authtime).

+ Expiration time of this key - optional (endtime): This field is present if the SKDC returns a public key pair only. If a certificate accompanies the public key pair, then this field is absent.

+ Service principal permissions - optional (sppac).

+ Transited realms - optional (transited).

+ Key data (keydata): In a PKP-REP message, the key-data structure in the miniticket contains only the public key of the client. The private key must not be included. If the client had also requested a new certificate, the certificate is included here. See section Section 2.6 for the key-data structure.

o Receipt (receipt): This is the part of the PKP-REP message that is encrypted by the SKDC to the client, using the secret key that the SKDC shares pair-wise with the client. It contains the following:

* Receipts flags - optional (tflags).

* Issuing SKDC’s realm (skdcrealm).

* Issuing SKDC’s identity (skdcname).

* Service principal’s realm - optional (sprealm).

* Service Principal’s identity (spname).

* Service Principal’s SKDC - optional (spskdc).

* Client permissions - optional (cpac).

* Time of authentication (authtime): This is the time of the creation of this receipt.

* Nonce from the Client’s previous PSK-REQ request message (nonce).

* Expiration time of this key (endtime).

* Key data (keydata): In a PKP-REP message, the key-data structure in the receipt contains both the public key and private key belonging to the requesting client. If the client had also requested a new certificate, the certificate is included here. See section Section 2.6 for the key-data structure.

o Extensions - optional (ext): Reserved for future extensions. This field is optional.

8. JSON Message Format

TBD.

9. Encryption and Checksums

TBD.

10. Security Considerations

TBD.

11. Privacy Considerations

TBD.

12. IANA Considerations

TBD.

13. Acknowledgments

We thank Jesse Walker for design inputs and initial review.

14. References

14.1. Normative References

[JSON] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", March 2014, <https://tools.ietf.org/html/rfc7159>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6347] Rescorla, E., "Datagram Transport Layer Security Version 1.2", January 2012, <http://tools.ietf.org/html/rfc6347>.

[RFC7252] Shelby, Z., "The Constrained Application Protocol (CoAP)", June 2014, <http://tools.ietf.org/html/rfc7252>.

14.2. Informative References

[ACE] Seitz, L., Ed., "ACE Use Cases", October 2012, <https://tools.ietf.org/wg/ace/draft-ietf-ace-usecases/>.

[BR-3KPD] Bellare, M. and P. Rogaway, "Entity Authentication and Key Distribution (In Advances in Cryptology, pages 110-125. Springer-Verlag, 1993)", September 1993, <http://link.springer.com/>.

[Choo04] Choo, K., Boyd, C., Hitchcock, Y., and G. Maitland, "On Session Identifiers in Provably Secure Protocols (Security in Communication Networks 4th International Conference, SCN 2004)", September 2004, <http://link.springer.com/>.

[Choo06] Choo, R., "Key Establishment: Proofs and Refutations", May 2006, <http://eprints.qut.edu.au/16262/1/Kim-Kwang_Choo_Thesis.pdf>.

[EPID] Brickell, E. and J. Li, "Enhanced Privacy ID (in NIST Privacy Enhancing Cryptography Conference 2011)", December 2011, <http://csrc.nist.gov/groups/ST/PEC2011/presentations2011/brickell.pdf>.

[MSKILE] Microsoft, ., "Kerberos Protocol Extensions (v20140502)", May 2014, <https://msdn.microsoft.com/en-us/library/cc233855.aspx>.

[MSPAC] Microsoft, ., "Privilege Attribute Certificate Data Structure (v20140502)", May 2014, <https://msdn.microsoft.com/en-us/library/cc237917.aspx>.

[NS] Needham, R. and M. Schroeder, "Using encryption for authentication in large networks of computers (CACM)", December 1978, <http://en.wikipedia.org/wiki/Needham-Schroeder_protocol>.

[OAuth2] Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, <http://tools.ietf.org/html/rfc6749>.

[OIDC] Sakimura, N., "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos V5", February 2005, <http://tools.ietf.org/html/rfc3961>.

[RFC3986] Berners-Lee, T., "Uniform Resource Identifier (URI): Generic Syntax", January 2005, <http://www.ietf.org/rfc/rfc3986.txt>.

[RFC4120] Neuman, C., "The Kerberos Network Authentication Service (V5)", July 2005, <http://tools.ietf.org/html/rfc4120>.

[RFC6113] Hartman, S., "A Generalized Framework for Kerberos Pre-Authentication", April 2011, <http://tools.ietf.org/html/rfc6113>.

[TPM] TCG, ., "Trusted Platform Module (TPM) Main Specification Level 2 Version 1.2", March 2011, <http://www.trustedcomputinggroup.org/resources/tpm_main_specification>.

[UMACORE] Hardjono, T., Ed., "User-Managed Access (UMA) Profile of OAuth 2.0", November 2014, <https://docs.kantarainitiative.org/uma/draft-uma-core.html>.

Appendix A. Document History

NOTE: To be removed by RFC editor before publication as an RFC.

Authors' Addresses

Thomas Hardjono
MIT

Email: hardjono@mit.edu

Ned Smith
Intel Corp

Email: ned.smith@intel.com

ACE Working Group
Internet-Draft
Intended status: Informational
Expires: April 25, 2016

L. Seitz, Ed.
SICS Swedish ICT AB
S. Gerdes, Ed.
Universitaet Bremen TZI
G. Selander
Ericsson
M. Mani
Itron
S. Kumar
Philips Research
October 23, 2015

Use Cases for Authentication and Authorization in Constrained Environments
draft-ietf-ace-usecases-10

Abstract

Constrained devices are nodes with limited processing power, storage space and transmission capacities. These devices in many cases do not provide user interfaces and are often intended to interact without human intervention.

This document includes a collection of representative use cases for authentication and authorization in constrained environments. These use cases aim at identifying authorization problems that arise during the lifecycle of a constrained device and are intended to provide a guideline for developing a comprehensive authentication and authorization solution for this class of scenarios.

Where specific details are relevant, it is assumed that the devices use the Constrained Application Protocol (CoAP) as communication protocol, however most conclusions apply generally.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Constrained devices [RFC7228] are nodes with limited processing power, storage space and transmission capacities. These devices are often battery-powered and in many cases do not provide user interfaces.

Constrained devices benefit from being interconnected using Internet protocols. However, deploying common security protocols can sometimes be difficult because of device or network limitations. Regardless, adequate security mechanisms are required to protect these constrained devices, which are expected to be integrated in all aspects of everyday life, from attackers wishing to gain control over the device's data or functions.

This document comprises a collection of representative use cases for the application of authentication and authorization in constrained environments. These use cases aim at identifying authorization problems that arise during the lifecycle of a constrained device. Note that this document does not aim at collecting all possible use cases.

We assume that the communication between the devices is based on the Representational State Transfer (REST) architectural style, i.e. a device acts as a server that offers resources such as sensor data and actuators. The resources can be accessed by clients, sometimes without human intervention (M2M). In some situations the communication will happen through intermediaries (e.g. gateways, proxies).

Where specific detail is necessary it is assumed that the devices communicate using CoAP [RFC7252], although most conclusions are generic.

1.1. Terminology

Readers are required to be familiar with the terms defined in [RFC7228].

2. Use Cases

This section includes the use cases; each use case first presents a general description of the application environment, than one or more specific use cases, and finally a summary of the authorization-related problems to be solved. The document aims at listing the relevant authorization problems and not to provide an exhaustive list. It might not be possible to address all of the listed problems with a single solution; There might be conflicting goals within or among some requirements.

There are various reasons for assigning a function (client or server) to a device, e.g. which device initiates the conversation, how do devices find each other, etc. The definition of the function of a device in a certain use case is not in scope of this document. Readers should be aware that there might be reasons for each setting and that endpoints might even have different functions at different times.

2.1. Container monitoring

The ability of sensors to communicate environmental data wirelessly opens up new application areas. Sensor systems make it possible to continuously track and transmit characteristics such as temperature, humidity and gas content while goods are transported and stored.

Sensors in this scenario have to be associated to the appropriate pallet of the respective container. Sensors as well as the goods belong to specific customers.

While in transit goods often pass stops where they are transloaded to other means of transportation, e.g. from ship transport to road transport.

Perishable goods need to be stored at constant temperature and with proper ventilation. Real-time information on the state of the goods is needed by both the transporter and the vendor. Transporters want to prioritize good that will expire soon. Vendors want to react when goods are spoiled to continue to fulfill delivery obligations.

The Intelligent Container (http://www.intelligentcontainer.com) is an example project that explores solutions to continuously monitor perishable goods.

2.1.1. Bananas for Munich

A fruit vendor grows bananas in Costa Rica for the German market. It instructs a transport company to deliver the goods via ship to Rotterdam where they are picked up by trucks and transported to a ripening facility. A Munich supermarket chain buys ripened bananas from the fruit vendor and transports them from the ripening facility to the individual markets with their own company trucks.

The fruit vendor's quality management wants to assure the quality of their products and thus equips the banana boxes with sensors. The state of the goods is monitored consistently during shipment and ripening and abnormal sensor values are recorded (U1.2). Additionally, the sensor values are used to control the climate within the cargo containers (U1.1, U1.5, U1.7). The sensors therefore need to communicate with the climate control system. Since a wrong sensor value leads to a wrong temperature and thus to spoiled goods, the integrity of the sensor data must be assured (U1.2, U1.3). The banana boxes within a container will in most cases belong to the same owner. Adjacent containers might contain goods and sensors of different owners (U1.1).

The personnel that transloads the goods must be able to locate the goods meant for a specific customer (U1.1, U1.6, U1.7). However the fruit vendor does not want to disclose sensor information pertaining to the condition of the goods to other companies and therefore wants to assure the confidentiality of this data (U1.4). Thus, the transloading personnel is only allowed to access logistic information (U1.1). Moreover, the transloading personnel is only allowed to access the data for the time of the transloading (U1.8).

Due to the high water content of the fruits, the propagation of radio waves is hindered, thus often inhibiting direct communication between nodes [Jedermann14]. Instead, messages are forwarded over multiple hops (U1.9). The sensors in the banana boxes cannot always reach the Internet during the journey (U1.10). Sensors may need to use relay stations owned by the transport company to connect to endpoints in the Internet.

In the ripening facility bananas are stored until they are ready to be sold. The banana box sensors are used to control the ventilation system and to monitor the degree of ripeness of the bananas. Ripe bananas need to be identified and sold before they spoil (U1.2, U1.8).

The supermarket chain gains ownership of the banana boxes when the bananas have ripened and are ready to leave the ripening facility.

2.1.2. Authorization Problems Summary

- U1.1 Fruit vendors and container owners want to grant different authorizations for their resources and/or endpoints to different parties.
- U1.2 The fruit vendor requires the integrity and authenticity of the sensor data that pertains the state of the goods for climate control and to ensure the quality of the monitored recordings.
- U1.3 The container owner requires the integrity and authenticity of the sensor data that is used for climate control.
- U1.4 The fruit vendor requires the confidentiality of the sensor data that pertains the state of the goods and the confidentiality of location data, e.g., to protect them from targeted attacks from competitors.
- U1.5 The fruit vendor may need different protection for several different types of data on the same endpoint, e.g., sensor data and the data used for logistics.
- U1.6 The fruit vendor and the transloading personnel require the authenticity and integrity of the data that is used to locate the goods, in order to ensure that the goods are correctly treated and delivered.
- U1.7 The container owner and the fruit vendor may not be present at the time of access and cannot manually intervene in the authorization process.
- U1.8 The fruit vendor, container owner and transloading company want to grant temporary access permissions to a party, in order to avoid giving permanent access to parties that are no longer involved in processing the bananas.
- U1.9 The fruit vendor, container owner and transloading company want their security objectives to be achieved, even if the messages between the endpoints need to be forwarded over multiple hops.
- U1.10 The constrained devices might not always be able to reach the Internet but still need to enact the authorization policies of their principals.

- o U1.11 Fruit vendors and container owners want to be able to revoke authorization on a malfunctioning sensor.

2.2. Home Automation

One application of the Internet of Things is home automation systems. Such a system can connect household devices that control, for example heating, ventilation, lighting, home entertainment, and home security to the Internet making them remotely accessible and manageable.

Such a system needs to accommodate a number of regular users (inhabitants, close friends, cleaning personnel) as well as a heterogeneous group of dynamically varying users (visitors, repairmen, delivery men).

As the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users.

2.2.1. Controlling the Smart Home Infrastructure

Alice and Bob own a flat which is equipped with home automation devices such as HVAC and shutter control, and they have a motion sensor in the corridor which controls the light bulbs there (U2.5).

Alice and Bob can control the shutters and the temperature in each room using either wall-mounted touch panels or an internet connected device (e.g. a smartphone). Since Alice and Bob both have a full-time job, they want to be able to change settings remotely, e.g. turn up the heating on a cold day if they will be home earlier than expected (U2.5).

The couple does not want people in radio range of their devices, e.g. their neighbors, to be able to control them without authorization. Moreover, they don’t want burglars to be able to deduce behavioral patterns from eavesdropping on the network (U2.8).

2.2.2. Seamless Authorization

Alice buys a new light bulb for the corridor and integrates it into the home network, i.e. makes resources known to other devices in the network. Alice makes sure that the new light bulb and her other devices in the network get to know the authorization policies for the new device. Bob is not at home, but Alice wants him to be able to control the new device with his devices (e.g. his smartphone) without the need for additional administration effort (U2.7). She provides the necessary configurations for that (U2.9, U2.10).

2.2.3. Remotely letting in a visitor

Alice and Bob have equipped their home with automated connected door-locks and an alarm system at the door and the windows. The couple can control this system remotely.

Alice and Bob have invited Alice's parents over for dinner, but are stuck in traffic and cannot arrive in time, while Alice's parents who use the subway will arrive punctually. Alice calls her parents and offers to let them in remotely, so they can make themselves comfortable while waiting (U2.1, U2.6). Then Alice sets temporary permissions that allow them to open the door, and shut down the alarm (U2.2). She wants these permissions to be only valid for the evening since she does not like it if her parents are able to enter the house as they see fit (U2.3, U2.4).

When Alice's parents arrive at Alice's and Bob's home, they use their smartphone to communicate with the door-lock and alarm system (U2.5, U2.9). The permissions Alice issued to her parents only allow limited access to the house (e.g. opening the door, turning on the lights). Certain other functions, such as checking the footage from the surveillance cameras is not accessible to them (U2.3).

Alice and Bob also issue similarly restricted permissions to e.g. cleaners, repairmen or their nanny (U2.3).

2.2.4. Selling the house

Alice and Bob have to move because Alice is starting a new job. They therefore decide to sell the house, and transfer control of all automated services to the new owners (U2.11). Before doing that they want to erase privacy relevant data from the logs of the automated systems, while the new owner is interested to keep some historic data e.g. pertaining to the behavior of the heating system (U2.12). At the time of transfer of the house, the new owners also wants make sure that permissions issued by the previous owners to access the house or connected devices (in the case where device management may have separate permissions from house access) are no longer valid (U2.13).

2.2.5. Authorization Problems Summary

- U2.1 A home owner (Alice and Bob in the example above) wants to spontaneously provision authorization means to visitors.

- U2.2 A home owner wants to spontaneously change the home's access control policies.

- U2.3 A home owner wants to apply different access rights for different users (including other inhabitants).
- U2.4 The home owners want to grant access permissions to a someone during a specified time frame.
- U2.5 The smart home devices need to be able to securely communicate with different control devices (e.g. wall-mounted touch panels, smartphones, electronic key fobs, device gateways).
- U2.6 The home owner wants to be able to configure authorization policies remotely.
- U2.7 Authorized Users want to be able to obtain access with little effort.
- U2.8 The owners of the automated home want to prevent unauthorized entities from being able to deduce behavioral profiles from devices in the home network.
- U2.9 Usability is particularly important in this scenario since the necessary authorization related tasks in the lifecycle of the device (commissioning, operation, maintenance and decommissioning) likely need to be performed by the home owners who in most cases have little knowledge of security.
- U2.10 Home Owners want their devices to seamlessly (and in some cases even unnoticeably) fulfill their purpose. Therefore the authorization administration effort needs to be kept at a minimum.
- U2.11 Home Owners want to be able to transfer ownership of their automated systems when they sell the house.
- U2.12 Home Owners want to be able to sanitize the logs of the automated systems, when transferring ownership, without deleting important operational data.
- U2.13 When a transfer of ownership occurs, the new owner wants to make sure that access rights created by the previous owner are no longer valid.

2.3. Personal Health Monitoring

Personal health monitoring devices, i.e. eHealth devices, are typically battery driven and located physically on or in the user to monitor some bodily function, such as temperature, blood pressure, or pulse rate. These devices typically connect to the Internet through an intermediary base-station, using wireless technologies and through

this connection they report the monitored data to some entity, which may either be the user, or a medical caregiver.

Medical data has always been considered as very sensitive, and therefore requires good protection against unauthorized disclosure. A frequent, conflicting requirement is the capability for medical personnel to gain emergency access, even if no specific access rights exist. As a result, the importance of secure audit logs increases in such scenarios.

Since the users are not typically trained in security (or even computer use), the configuration must use secure default settings, and the interface must be well adapted to novice users. Parts of the system must operate with minimal maintenance. Especially frequent changes of battery are unacceptable.

There is a plethora of wearable health monitoring technology and the need for open industry standards to ensure interoperability between products has lead to initiatives such as Continua Alliance (continuaalliance.org) and Personal Connected Health Alliance (pchalliance.org).

2.3.1. John and the heart rate monitor

John has a heart condition, that can result in sudden cardiac arrests. He therefore uses a device called HeartGuard that monitors his heart rate and his location (U3.7). In case of a cardiac arrest it automatically sends an alarm to an emergency service, transmitting John’s current location (U3.1). Either the device has long range connectivity itself (e.g. via GSM) or it uses some intermediary, nearby device (e.g. John’s smartphone) to transmit such an alarm. To ensure Johns safety, the device is expected to be in constant operation (U3.3, U3.6).

The device includes an authentication mechanism, in order to prevent other persons who get physical access to it from acting as the owner and altering the access control and security settings (U3.8).

John can configure additional persons that get notified in an emergency, for example his daughter Jill. Furthermore the device stores data on John’s heart rate, which can later be accessed by a physician to assess the condition of John’s heart (U3.2).

However John is a privacy conscious person, and is worried that Jill might use HeartGuard to monitor his location while there is no emergency. Furthermore he doesn't want his health insurance to get access to the HeartGuard data, or even to the fact that he is wearing a HeartGuard, since they might refuse to renew his insurance if they decided he was too big a risk for them (U3.8).

Finally John, while being comfortable with modern technology and able to operate it reasonably well, is not trained in computer security. He therefore needs an interface for the configuration of the HeartGuard security that is easy to understand and use (U3.5). If John does not understand the meaning of a setting, he tends to leave it alone, assuming that the manufacturer has initialized the device to secure settings (U3.4).

NOTE: Monitoring of some state parameter (e.g. an alarm button) and the position of a person also fits well into an elderly care service. This is particularly useful for people suffering from dementia, where the relatives or caregivers need to be notified of the whereabouts of the person under certain conditions. In this case it is not the patient that decides about access.

2.3.2. Authorization Problems Summary

- U3.1 The wearer of an eHealth device (John in the example above) wants to pre-configure special access rights in the context of an emergency.
- U3.2 The wearer of an eHealth device wants to selectively allow different persons or groups access to medical data.
- U3.3 Battery changes are very inconvenient and sometimes impractical, so battery life impacts of the authorization mechanisms need to be minimized.
- U3.4 Devices are often used with default access control settings which might threaten the security objectives of the device's users.
- U3.5 Wearers of eHealth devices are often not trained in computer use, and especially computer security.
- U3.6 Security mechanisms themselves could provide opportunities for denial of service attacks, especially on the constrained devices.
- U3.7 The device provides a service that can be fatal for the wearer if it fails. Accordingly, the wearer wants the device to

have a high degree of resistance against attacks that may cause the device to fail to operate partially or completely.

- o U3.8 The wearer of an eHealth device requires the integrity and confidentiality of the data measured by the device.

2.4. Building Automation

Buildings for commercial use such as shopping malls or office buildings nowadays are equipped increasingly with semi-automatic components to enhance the overall living quality and to save energy where possible. This includes for example heating, ventilation and air condition (HVAC) as well as illumination and security systems such as fire alarms. These components are being increasingly managed centrally in a Building and Lighting Management System (BLMS) by a facility manager.

Different areas of these buildings are often exclusively leased to different companies. However they also share some of the common areas of the building. Accordingly, a company must be able to control the lighting and HVAC system of its own part of the building and must not have access to control rooms that belong to other companies.

Some parts of the building automation system such as entrance illumination and fire alarm systems are controlled either by all parties together or by a facility management company.

2.4.1. Device Lifecycle

2.4.1.1. Installation and Commissioning

Installation of the building automation components often start even before the construction work is completed. Lighting is one of the first components to be installed in new buildings. A lighting plan created by a lighting designer provides the necessary information related to the kind of lighting devices (luminaires, sensors and switches) to be installed along with their expected behavior. The physical installation of the correct lighting devices at the right locations are done by electricians based on the lighting plan. They ensure that the electrical wiring is performed according to local regulations and lighting devices which may be from multiple manufacturers are connected to the electrical power supply properly. After the installation, lighting can be used in a default out-of-box mode for e.g. at full brightness when powered on. After this step (or in parallel in a different section of the building), a lighting commissioner adds the devices to the building domain (U4.1) and performs the proper configuration of the lights as prescribed in the

lighting plan. This involves for example grouping to ensure that light points react together, more or less synchronously (U4.8) and defining lighting scenes for particular areas of the building. The commissioning is often done in phases, either by one or more commissioners, on different floors. The building lighting network at this stage may be in different network islands with no connectivity between them due to lack of the IT infrastructure.

After this, other building components like HVAC and security systems are similarly installed by electricians and later commissioned by their respective domain professionals. Similar configurations related to grouping (U4.8) are required to ensure for e.g. HVAC equipment are controlled by the closest temperature sensor.

For the building IT systems, the Ethernet wiring is initially laid out in the building according to the IT plan. The IT network is commissioned often after the construction is completed to avoid any damage to sensitive networking and computing equipment. The commissioning is performed by an IT engineer with additional switches (wired and/or wireless), IP routers and computing devices. Direct Internet connectivity for all installed/commissioned devices in the building is only available at this point. The BLMS that monitors and controls the various building automation components are only connected to the field devices at this stage. The different network islands (for lighting and HVAC) are also joined together without any further involvement of domain specialist such as lighting or HVAC commissioners.

2.4.1.2. Operational

The building automation systems is now finally ready and the operational access is transferred to the facility management company of the building (U4.2). The facility manager is responsible for monitoring and ensuring that the building automation systems meets the needs of the building occupants. If changes are needed, the facility management company hires an external installation and commissioning company to perform the changes.

Different parts of the building are rented out to different companies for office space.
The tenants are provided access to use the automated HVAC, lighting and physical access control systems deployed. The safety of the occupants are also managed using automated systems, such as a fire alarm system, which is triggered by several smoke detectors which are spread out across the building.

Company A’s staff move into the newly furnished office space. Most lighting is controlled by presence sensors which control the lighting

of specific group of lights based on the authorization rules in the BLMS. Additionally employees are allowed to manually override the lighting brightness and color in their office rooms by using the switches or handheld controllers. Such changes are allowed only if the authorization rules exist in the BLMS. For example lighting in the corridors may not be manually adjustable.

At the end of the day, lighting is dimmed down or switched off if no occupancy is detected even if manually overridden during the day.

On a later date company B also moves into the same building, and shares some of the common spaces and associated building automation components with company A (U4.2, U4.9).

2.4.1.3. Maintenance

Company A’s staff are annoyed that the lighting switches off too often in their rooms if they work silently in front of their computer. Company A notifies the the facility manager of the building to increase the delay before lights switch off. The facility manager can either configure the new values directly in the BLMS or if additional changes are needed on the field devices, hires a commissioning Company C to perform the needed changes (U4.4).

Company C gets the necessary authorization from the facility management company to interact with the BLMS. The commissioner’s tool gets the necessary authorization from BLMS to send a configuration change to all lighting devices in Company A’s offices to increase their delay before they switch off.

At some point the facility management company wants to update the firmware of lighting devices in order to eliminate software bugs. Before accepting the new firmware, each device checks the authorization of the facility management company to perform this update (U4.13).

A network diagnostic tool of the BLMS detects that a luminaire in one of the Company A’s office room is no longer connected to the network. The BLMS alerts the facility manager to replace the luminaire. The facility manager replaces the old broken luminaire and informs the BLMS of the identity (for e.g. MAC address) of the newly added device. The BLMS then authorizes the new device onto the system and transfers seamlessly all the permissions of the previous broken device to the replacement device (U4.12).

2.4.1.4. Recommissioning

A vacant area of the building has been recently leased to company A. Before moving into its new office, Company A wishes to replace the lighting with a more energy efficient and a better light quality luminaries. They hire an installation and commissioning company C to redo the illumination. Company C is instructed to integrate the new lighting devices, which may be from multiple manufacturers, into the existing lighting infrastructure of the building which includes presence sensors, switches, controllers etc (U4.1).

Company C gets the necessary authorization from the facility management company to interact with the existing BLMS (U4.4). To prevent disturbance to other occupants of the building, Company C is provided authorization to perform the commissioning only during non-office hours and only to modify configuration on devices belonging to the domain of Company A's space (U4.5). Before removing existing devices, all security and configuration material that belongs to the domain are deleted and the devices are set back to factory state (U4.3). This ensures that these devices may be reused at other installations or in other parts of the same building without affecting future operations. After installation (wiring) of the new lighting devices, the commissioner adds the devices into the company A's lighting domain.

Once the devices are in the correct domain, the commissioner authorizes the interaction rules between the new lighting devices and existing devices like presence sensors (U4.7). For this, the commissioner creates the authorization rules on the BLMS which define which lights form a group and which sensors/switches/controllers are allowed to control which groups (U4.8). These authorization rules may be context based like time of the day (office or non-office hours) or location of the handheld lighting controller etc (U4.5).

2.4.1.5. Decommissioning

Company A has noticed that the handheld controllers are often misplaced and hard to find when needed. So most of the time staff use the existing wall switches for manual control. Company A decides it would be better to completely remove handheld controllers and asks Company C to decommission them from the lighting system (U4.4).

Company C again gets the necessary authorization from the facility management company to interact with the BLMS. The commissioner now deletes any rules that allowed handheld controllers authorization to control the lighting (U4.3, U4.6). Additionally the commissioner instructs the BLMS to push these new rules to prevent cached rules at the end devices from being used. Any cryptographic key material belonging to the site in the handheld controllers are also removed and they are set to the factory state (U4.3).

2.4.2. Public Safety

The fire department requires that as part of the building safety code, that the building have sensors that sense the level of smoke, heat, etc., when a fire breaks out. These sensors report metrics which are then used by a back-end server to map safe areas and un-safe areas within a building and also possibly the structural integrity of the building before fire-fighters may enter it. Sensors may also be used to track where human/animal activity is within the building. This will allow people stuck within the building to be guided to safer areas and suggest possible actions that they may take (e.g. using a client application on their phones, or loudspeaker directions) in order to bring them to safety. In certain cases, other organizations such as the Police, Ambulance, and federal organizations are also involved and therefore the co-ordination of tasks between the various entities have to be carried out using efficient messaging and authorization mechanisms.

2.4.2.1. A fire breaks out

On a really hot day James who works for company A turns on the air condition in his office. Lucy who works for company B wants to make tea using an electric kettle. After she turned it on she goes outside to talk to a colleague until the water is boiling. Unfortunately, her kettle has a malfunction which causes overheating and results in a smoldering fire of the kettle’s plastic case.

Due to the smoke coming from the kettle the fire alarm is triggered. Alarm sirens throughout the building are switched on simultaneously (using a group communication scheme) to alert the staff of both companies (U4.8). Additionally, the ventilation system of the whole building is closed off to prevent the smoke from spreading and to withdraw oxygen from the fire. The smoke cannot get into James’ office although he turned on his air condition because the fire alarm overrides the manual setting by sending commands (using group communication) to switch off all the air conditioning (U4.10).

The fire department is notified of the fire automatically and arrives within a short time. They automatically get access to all parts of the building according to an emergency authorization policy (U4.4, U4.5). After inspecting the damage and extinguishing the smoldering fire a fire fighter resets the fire alarm because only the fire department is authorized to do that (U4.4, U4.11).

2.4.3. Authorization Problems Summary

- U4.1 During commissioning, the building owner or the companies add new devices to their administrative domain. Access control should then apply to these devices seamlessly.

- U4.2 During a handover, the building owner or the companies integrate devices that formerly belonged to a different administrative domain to their own administrative domain. Access control of the old domain should then cease to apply, with access control of the new domain taking over.

- U4.3 During decommissioning, the building owner or the companies remove devices from their administrative domain. Access control should cease to apply to these devices and relevant credentials need to be erased from the devices.

- U4.4 The building owner and the companies want to be able to delegate specific access rights for their devices to others.

- U4.5 The building owner and the companies want to be able to define context-based authorization rules.

- U4.6 The building owner and the companies want to be able to revoke granted permissions and delegations.

- U4.7 The building owner and the companies want to allow authorized entities to send data to their endpoints (default deny).

- U4.8 The building owner and the companies want to be able to authorize a device to control several devices at the same time using a group communication scheme.

- U4.9 The companies want to be able to interconnect their own subsystems with those from a different operational domain while keeping the control over the authorizations (e.g. granting and revoking permissions) for their endpoints and devices.

- U4.10 The authorization mechanisms must be able to cope with extremely time-sensitive operations which have to be carried out in a quick manner.

- U4.11 The building owner and the public safety authorities want to be able to perform data origin authentication on messages sent and received by some of the systems in the building.

- U4.12 The building owner should be allowed to replace an existing device with a new device providing the same functionality within their administrative domain. Access control from the replaced device should then apply to these new devices seamlessly.

- o U4.13 When software on a device is updated, this update needs to be authenticated and authorized.

2.5. Smart Metering

Automated measuring of customer consumption is an established technology for electricity, water, and gas providers. Increasingly these systems also feature networking capability to allow for remote management. Such systems are in use for commercial, industrial and residential customers and require a certain level of security, in order to avoid economic loss to the providers, vulnerability of the distribution system, as well as disruption of services for the customers.

The smart metering equipment for gas and water solutions is battery driven and communication should be used sparingly due to battery consumption. Therefore the types of meters sleep most of the time, and only wake up every minute/hour to check for incoming instructions. Furthermore they wake up a few times a day (based on their configuration) to upload their measured metering data.

Different networking topologies exist for smart metering solutions. Based on environment, regulatory rules and expected cost, one or a mixture of these topologies may be deployed to collect the metering information. Drive-By metering is one of the most current solutions deployed for collection of gas and water meters.

Various stakeholders have a claim on the metering data. Utility companies need the data for accounting, the metering equipment may be operated by a third party Service Operator who needs to maintain it, and the equipment is installed in the premises of the consumers, measuring their consumption, which entails privacy questions.

2.5.1. Drive-by metering

A service operator offers smart metering infrastructures and related services to various utility companies. Among these is a water provider, who in turn supplies several residential complexes in a city. The smart meters are installed in the end customer’s homes to measure water consumption and thus generate billing data for the utility company, they can also be used to shut off the water if the bills are not paid (U5.1, U5.3). The meters do so by sending and receiving data to and from a base station (U5.2). Several base stations are installed around the city to collect the metering data. However in the denser urban areas, the base stations would have to be installed very close to the meters. This would require a high number of base stations and expose this more expensive equipment to manipulation or sabotage. The service operator has therefore chosen

another approach, which is to drive around with a mobile base-station and let the meters connect to that in regular intervals in order to gather metering data (U5.4, U5.6, U5.8).

2.5.2. Meshed Topology

In another deployment, the water meters are installed in a building that already has power meters installed, the latter are mains powered, and are therefore not subject to the same power saving restrictions. The water meters can therefore use the power meters as proxies, in order to achieve better connectivity. This requires the security measures on the water meters to work through intermediaries (U5.9).

2.5.3. Advanced Metering Infrastructure

A utility company is updating its old utility distribution network with advanced meters and new communication systems, known as an Advanced Metering Infrastructure (AMI). AMI refers to a system that measures, collects and analyzes usage, and interacts with metering devices such as electricity meters, gas meters, heat meters, and water meters, through various communication media either on request (on-demand) or on pre-defined schedules. Based on this technology, new services make it possible for consumers to control their utility consumption (U5.2, U5.7) and reduce costs by supporting new tariff models from utility companies, and more accurate and timely billing. However the end-consumers do not want unauthorized persons to gain access to this data. Furthermore, the fine-grained measurement of consumption data may induce privacy concerns, since it may allow others to create behavioral profiles (U5.5, U5.10).

The technical solution is based on levels of data aggregation between smart meters located at the consumer premises and the Meter Data Management (MDM) system located at the utility company (U5.9). For reasons of efficiency and cost, end-to-end connectivity is not always feasible, so metering data is stored and aggregated in various intermediate devices before being forwarded to the utility company, and in turn accessed by the MDM. The intermediate devices may be operated by a third party service operator on behalf of the utility company (U5.7). One responsibility of the service operator is to make sure that meter readings are performed and delivered in a regular, timely manner. An example of a Service Level Agreement between the service operator and the utility company is e.g. "at least 95 % of the meters have readings recorded during the last 72 hours".

2.5.4. Authorization Problems Summary

- U5.1 Devices are installed in hostile environments where they are physically accessible by attackers (including dishonest customers). The service operator and the utility company want to make sure that an attacker cannot use data from a captured device to attack other parts of their infrastructure.

- U5.2 The utility company wants to control which entities are allowed to send data to, and read data from their endpoints.

- U5.3 The utility company wants to ensure the integrity of the data stored on their endpoints.

- U5.4 The utility company wants to protect such data transfers to and from their endpoints.

- U5.5 Consumers want to access their own usage information and also prevent unauthorized access by others.

- U5.6 The devices may have intermittent Internet connectivity but still need to enact the authorization policies of their principals.

- U5.7 Neither the service operator nor the utility company are always present at the time of access and cannot manually intervene in the authorization process.

- U5.8 When authorization policies are updated it is impossible, or at least very inefficient to contact all affected endpoints directly.

- U5.9 Authorization and authentication must work even if messages between endpoints are stored and forwarded over multiple nodes.

- U5.10 Consumers may not want the Service Operator, the Utility company or others to have access to a fine-grained level of consumption data that allows the creation of behavioral profiles.

2.6. Sports and Entertainment

In the area of leisure time activities, applications can benefit from the small size and weight of constrained devices. Sensors and actuators with various functions can be integrated into fitness equipment, games and even clothes. Users can carry their devices around with them at all times.

Usability is especially important in this area since users will often want to spontaneously interconnect their devices with others. Therefore the configuration of access permissions must be simple and fast and not require much effort at the time of access.

Continuously monitoring allows authorized users to create behavioral or movement profiles, which corresponds on the devices intended use, and unauthorized access to the collected data would allow an attacker to create the same profiles.
Moreover, the aggregation of data can seriously increase the impact on the privacy of the users.

2.6.1. Dynamically Connecting Smart Sports Equipment

Jody is a an enthusiastic runner. To keep track of her training progress, she has smart running shoes that measure the pressure at various points beneath her feet to count her steps, detect irregularities in her stride and help her to improve her posture and running style. On a sunny afternoon, she goes to the Finnbahn track near her home to work out. She meets her friend Lynn who shows her the smart fitness watch she bought a few days ago. The watch can measure the wearer’s pulse, show speed and distance, and keep track of the configured training program. The girls detect that the watch can be connected with Jody’s shoes and then can additionally display the information the shoes provide.

Jody asks Lynn to let her try the watch and lend it to her for the afternoon. Lynn agrees but doesn’t want Jody to access her training plan (U6.4). She configures the access policies for the watch so that Jody’s shoes are allowed to access the display and measuring features but cannot read or add training data (U6.1, U6.2). Jody’s shoes connect to Lynn’s watch after only a press of a button because Jody already configured access rights for devices that belong to Lynn a while ago (U6.3). Jody wants the device to report the data back to her fitness account while she borrows it, so she allows it to access her account temporarily.

After an hour, Jody gives the watch back and both girls terminate the connection between their devices.

2.6.2. Authorization Problems Summary

- U6.1 Sports equipment owners want to be able to grant access rights dynamically when needed.

- U6.2 Sports equipment owners want the configuration of access rights to work with very little effort.

- o U6.3 Sports equipment owners want to be able to pre-configure access policies that grant certain access permissions to endpoints with certain attributes (e.g. endpoints of a certain user) without additional configuration effort at the time of access.

- o U6.4 Sports equipment owners want to protect the confidentiality of their data for privacy reasons.

2.7. Industrial Control Systems

Industrial control systems (ICS) and especially supervisory control and data acquisition systems (SCADA) use a multitude of sensors and actuators in order to monitor and control industrial processes in the physical world. Example processes include manufacturing, power generation, and refining of raw materials.

Since the advent of the Stuxnet worm it has become obvious to the general public how vulnerable these kind of systems are, especially when connected to the Internet [Karnouskos11]. The severity of these vulnerabilities are exacerbated by the fact that many ICS are used to control critical public infrastructure, such as nuclear power, water treatment of traffic control. Nevertheless the economical advantages of connecting such systems to the Internet can be significant if appropriate security measures are put in place (U7.5).

2.7.1. Oil Platform Control

An oil platform uses an industrial control system to monitor data and control equipment. The purpose of this system is to gather and process data from a large number of sensors, and control actuators such as valves and switches to steer the oil extraction process on the platform. Raw data, alarms, reports and other information are also available to the operators, who can intervene with manual commands. Many of the sensors are connected to the controlling units by direct wire, but the operator is slowly replacing these units by wireless ones, since this makes maintenance easier (U7.4).

Some of the controlling units are connected to the Internet, to allow for remote administration, since it is expensive and inconvenient to fly in a technician to the platform (U7.3).

The main interest of the operator is to ensure the integrity of control messages and sensor readings (U7.1). Access in some cases needs to be restricted, e.g. the operator wants wireless actuators only to accept commands by authorized control units (U7.2).

The owner of the platform also wants to collect auditing information for liability reasons (U7.1).

Different levels of access apply e.g. for regular operators, vs. maintenance technician, vs. auditors of the platform (U7.6)

2.7.2. Authorization Problems Summary

- U7.1 The operator of the platform wants to ensure the integrity and confidentiality of sensor and actuator data.
- U7.2 The operator wants to ensure that data coming from sensors and commands sent to actuators are authentic.
- U7.3 Some devices do not have direct Internet connection, but still need to implement current authorization policies.
- U7.4 Devices need to authenticate the controlling units, especially those using a wireless connection.
- U7.5 The execution of unauthorized commands or the failure to execute an authorized command in an ICS can lead to significant financial damage, and threaten the availability of critical infrastructure services. Accordingly, the operator wants a authentication and authorization mechanisms that provide a very high level of security.
- U7.6 Different users should have different levels of access to the control system (e.g. operator vs. auditor).

3. Security Considerations

As the use cases listed in this document demonstrate, constrained devices are used in various environments. These devices are small and inexpensive and this makes it easy to integrate them into many aspects of everyday life. With access to vast amounts of valuable data and possibly control of important functions these devices need to be protected from unauthorized access. Protecting seemingly innocuous data and functions will lessen the possible effects of aggregation; attackers collecting data or functions from several sources can gain insights or a level of control not immediately obvious from each of these sources on its own.

Not only the data on the constrained devices themselves is threatened, the devices might also be abused as an intrusion point to infiltrate a network. Once an attacker gains control over the device, it can be used to attack other devices as well. Due to their limited capabilities, constrained devices appear as the weakest link in the network and hence pose an attractive target for attackers.

This section summarizes the security problems highlighted by the use cases above and provides guidelines for the design of protocols for authentication and authorization in constrained RESTful environments.

3.1. Attacks

This document lists security problems that users of constrained devices want to solve. Further analysis of attack scenarios is not in scope of the document. However, there are attacks that must be considered by solution developers.

Because of the expected large number of devices and their ubiquity, constrained devices increase the danger from Pervasive Monitoring [RFC7258] attacks. Solution Designers should consider this in the design of their security solution and provide for protection against this type of attack. In particular, messages containing sensitive data that are send over unprotected channels should be encrypted if possible.

Attacks aimed at altering data in transit (e.g. to perpetrate fraud) are a problem that is addressed in many web security protocols such as TLS or IPSec. Developers need to consider this type of attacks, and make sure that the protection measures they implement are adapted to the constrained environment.

As some of the use cases indicate, constrained devices may be installed in hostile environments where they are physically accessible (see Section 2.5). Protection from physical attacks is not in the scope of this document, but should be kept in mind by developers of authorization solutions.

Denial of service (DoS) attacks threaten the availability of services a device provides and constrained devices are especially vulnerable to these types of attacks because of their limitations. Attackers can illicit a temporary or, if the battery is drained, permanent failure in a service simply by repeatedly flooding the device with connection attempts; for some services (see section Section 2.3), availability is especially important. Solution designers must be particularly careful to consider the following limitations in every part of the authorization solution:

- Battery usage
- Number of required message exchanges
- Size of data that is transmitted (e.g. authentication and access control data)

- Size of code required to run the protocols
- Size of RAM memory and stack required to run the protocols
- Resources blocked by partially completed exchanges (e.g. while one party is waiting for a transaction time to run out)

Solution developers also need to consider whether the session should be protected from information disclosure and tampering.

3.2. Configuration of Access Permissions

- The access control policies need to be enforced (all use cases): The information that is needed to implement the access control policies needs to be provided to the device that enforces the authorization and applied to every incoming request.
- A single resource might have different access rights for different requesting entities (all use cases).

  Rationale: In some cases different types of users need different access rights, as opposed to a binary approach where the same access permissions are granted to all authenticated users.
- A device might host several resources where each resource has its own access control policy (all use cases).
- The device that makes the policy decisions should be able to evaluate context-based permissions such as location or time of access (see Section 2.2, Section 2.3, Section 2.4). Access may depend on local conditions, e.g. access to health data in an emergency. The device that makes the policy decisions should be able to take such conditions into account.

3.3. Authorization Considerations

- Devices need to be enabled to enforce authorization policies without human intervention at the time of the access request (see Section 2.1, Section 2.2, Section 2.4, Section 2.5).
- Authorization solutions need to consider that constrained devices might not have internet access at the time of the access request (see Section 2.1, Section 2.3, Section 2.5, Section 2.6).
- It should be possible to update access control policies without manually re-provisioning individual devices (see Section 2.2, Section 2.3, Section 2.5, Section 2.6).

Rationale: Peers can change rapidly which makes manual re-provisioning unreasonably expensive.

- Authorization policies may be defined to apply to a large number of devices that might only have intermittent connectivity. Distributing policy updates to every device for every update might not be a feasible solution (see Section 2.5).

- It must be possible to dynamically revoke authorizations (see e.g. Section 2.4).

- The authentication and access control protocol can put undue burden on the constrained system resources of a device participating in the protocol. An authorization solutions must take the limitations of the constrained devices into account (all use cases, see also Section 3.1).

- Secure default settings are needed for the initial state of the authentication and authorization protocols (all use cases).

  Rationale: Many attacks exploit insecure default settings, and experience shows that default settings are frequently left unchanged by the end users.

- Access to resources on other devices should only be permitted if a rule exists that explicitly allows this access (default deny) (see e.g. Section 2.4).

- Usability is important for all use cases. The configuration of authorization policies as well as the gaining access to devices must be simple for the users of the devices. Special care needs to be taken for scenarios where access control policies have to be configured by users that are typically not trained in security (see Section 2.2, Section 2.3, Section 2.6).

- Software updates are an important operation for which correct authorization is crucial. Additionally authenticating the receiver of a software update is also important, for example to make sure that the update has been received by the intended device.

3.4. Proxies

In some cases, the traffic between endpoints might go through intermediary nodes (e.g. proxies, gateways). This might affect the function or the security model of authentication and access control protocols e.g. end-to-end security between endpoints with DTLS might not be possible (see Section 2.5).

4. Privacy Considerations

The constrained devices in focus of this document collect data from the physical world via sensors or affect their surrounding via actuators. The collected and processed data often can be associated with individuals. Since sensor data may be collected and distributed on a regular interval a significant amount of information about an individual can be collected and used as input to learning algorithms as part of big data analysis and used in an automated decision making process.

Offering privacy protection for individuals is important to guarantee that only authorized entities are allowed to access collected data and to trigger actions, to obtain consent prior to the sharing of data, and to deal with other privacy-related threats outlined in RFC 6973.

RFC 6973 was written as guidance for engineers designing technical solutions. For a short description about the deployment-related aspects of privacy and further references relevant for the Internet of Things sector please read Section 7 of RFC 7452.

5. Acknowledgments

The authors would like to thank Olaf Bergmann, Sumit Singhal, John Mattson, Mohit Sethi, Carsten Bormann, Martin Murillo, Corinna Schmitt, Hannes Tschofenig, Erik Wahlstroem, Andreas Baeckman, Samuel Erdtman, Steve Moore, Thomas Hardjono, Kepeng Li, Jim Schaad, Prashant Jhingran, Kathleen Moriarty, and Sean Turner for reviewing and/or contributing to the document. Also, thanks to Markus Becker, Thomas Poetsch and Koojana Kuladinithi for their input on the container monitoring use case. Furthermore the authors thank Akbar Rahman, Chonggang Wang, Vinod Choyi, and Abhinav Somaraju who contributed to the building automation use case.

Ludwig Seitz and Goeran Selander worked on this document as part of EIT-ICT Labs activity PST-14056.

6. IANA Considerations

This document has no IANA actions.

7. Informative References

[Jedermann14]
Jedermann, R., Poetsch, T., and C. LLoyd, "Communication techniques and challenges for wireless food quality monitoring", Philosophical Transactions of the Royal

Society A Mathematical, Physical and Engineering Sciences, May 2014.

[Karnouskos11] Karnouskos, S., "Stuxnet Worm Impact on Industrial Cyber-Physical System Security", IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society, pp. 4490-4494 , November 2011.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

Authors' Addresses

Ludwig Seitz (editor)
SICS Swedish ICT AB
Scheelevaegen 17
Lund 223 70
Sweden

Email: ludwig@sics.se

Stefanie Gerdes (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

Phone: +49-421-218-63906
Email: gerdes@tzi.org

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
Sweden

Email: goran.selander@ericsson.com

Mehdi Mani
Itron
52, rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: Mehdi.Mani@itron.com

Sandeep S. Kumar
Philips Research
High Tech Campus
Eindhoven 5656 AA
The Netherlands

Email: sandeep.kumar@philips.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 10, 2015

H. Tschofenig
ARM Limited
E. Maler
Forgerock
E. Wahlstroem
S. Erdtman
Nexus Technology

March 9, 2015

Authentication and Authorization for Constrained Environments Using OAuth and UMA

draft-maler-ace-oauth-uma-00.txt

Abstract

Authentication and authorization are fundamental security features used in Internet and Web applications. Providing the same level of security functionality to the Internet of Things (IoT) environment as well is a logical enhancement and reduces the risk of unauthorized access to personal data.

IoT devices, however, have limitations in terms of processing power, memory, user interface, Internet connectivity, etc. Since many use cases span Web and IoT environments and the question of "Web" vs. "IoT" can in some cases be considered a continuum, it is required to find security solutions that can accommodate the capabilities and constraints of both environments without significant compromises.

Thus, an approach of adapting already standardized and deployed authentication and authorization technologies is worth examining. This document describes how the Web Authorization Protocol (OAuth) in combination with User-Managed Access (UMA) can be used for an IoT environment to bring Web-scale authorization services to the IoT world.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 10, 2015.

Table of Contents

1. Introduction

Deciding when a certain use case falls under the category of IoT and when it is not turns out to be a difficult task. For this reason, [RFC7228] made an attempt to describe characteristics of constrained-node networks and highlights some of the challenges. Companies often

have some degree of freedom to make trade-off decisions, for example, in terms of cost vs. physically available resources to push the boundaries of what can be done with IoT devices.

Manufacturers must take not only hardware costs into account, but also software development costs; reusing existing software, standards, practices, and expertise can help to lower the total cost of a product. Hence, the use cases combine the already existing identity and access management infrastructure with access control to objects in the physical world.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document leverages terminology from [RFC6749] and [I-D.hardjono-oauth-umacore] . Especially pertinent definitions are paraphrased below.

Resource Owner: An entity capable of granting access to a protected resource.

Resource Server: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

Authorization Server: The server issuing access tokens to the client after successfully authorizing it.

Requesting Party: An entity (which may or may not be the same as the resource owner) that uses a client to seek access to a protected resource.

Client: An application making protected resource requests with the resource owner’s authorization and on the requesting party’s behalf.

3. Use Cases

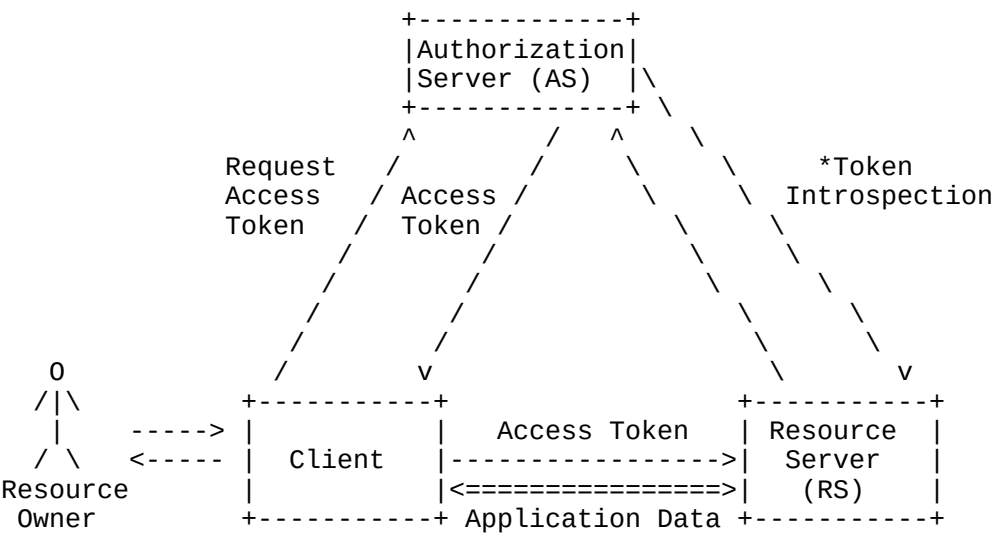
The sub-sections below illustrate some use cases that start with classic OAuth functionality and then extend it to functionality only available with UMA-based environments. The scenarios involve Web, smart phone app, and IoT devices. Unlike the scenarios described in [I-D.ietf-ace-usecases] this write-up is not solution agnostic but

instead aims to take the OAuth/UMA solutions into account. In a stepwise refinement we then add even more details in Section 5.

3.1. Using OAuth with Scales

In a classic OAuth flow, an end-user (the resource owner) can enable a client application to call an API (at the resource server) on his or her behalf securely and with authorized consent, without having to reveal his or her credentials, such as a username and password, to the client. An app-specific access token (issued by the authorization server at which the resource owner is able to authenticate), whose operation may be scoped to some subset of the API's capabilities, is substituted for the long-term credentials instead.

The basic OAuth architecture is shown in Figure 1 and the corresponding message exchange in Figure 2.



*: indicates optional exchange.

Figure 1: OAuth Architecture.

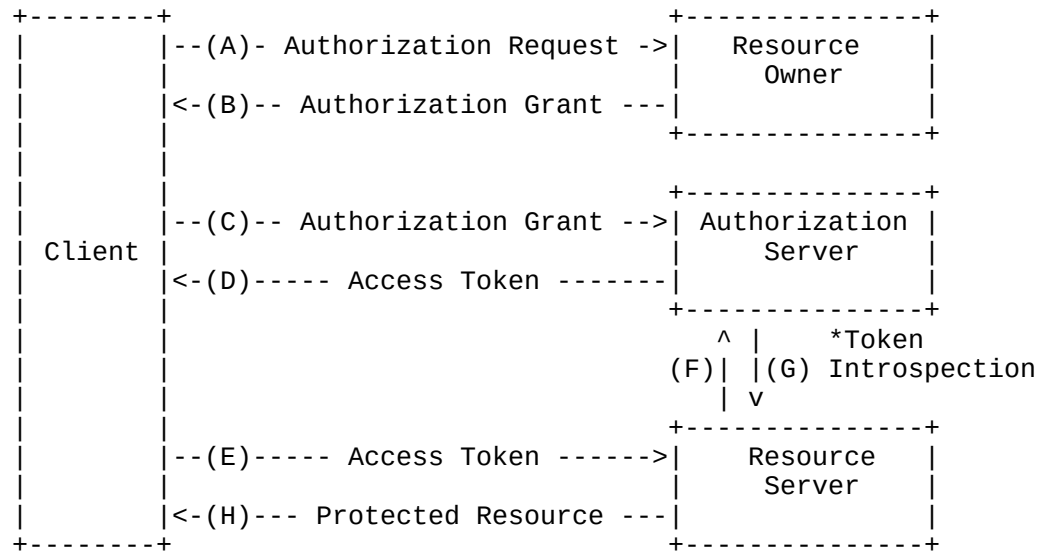


Figure 2: OAuth 2.0 Message Exchange.

We can apply a similar pattern to IoT devices as well. For example, envision an end-user Alice and her new purchase of an Internet-connected scale designed for "quantified self" scenarios. In our example, the scale has a micro-controller that was pre-provisioned with a certificate during manufacturing enabling the device to authenticate itself to the vendor-authorized software update server as well as to other parties. The identifier used for authentication of a scale is something as benign as an EUI-64 serial number.

Once the identifier used by the scale and Alice’s account information have been provisioned into an online repository, and if Alice can demonstrate appropriate control of the device -- for example, by entering a confirmable PIN code or serial number that was packaged with the shipped device into her online account record, whether through a Web or mobile app -- it is possible to treat the device as an OAuth client and issue it an OAuth token so that it can act on Alice’s behalf.

The value of this association is that any API calls made by the scale, for example to report Alice’s weight, body mass index (BMI), or progress against health goals into her online account, will be associated with her alone. If other household members use the scale as well, their unique associations will ensure that their data will go to the right place (assuming there is a mechanism at the scale

that allows family members to be differentiated). Further, each token can be revoked and expired exactly like any other OAuth token.

3.2. Using UMA with Scales

UMA builds on top of OAuth (and optionally OpenID Connect [OIDC]) to let an end-user achieve three main goals:

1. authorize other parties to access APIs under his or her control using client applications;

2. set conditions for access so that those other parties may have to provide "claims" and do step-up authentication to get access (in a so-called claims gathering process); and

3. centralize management of all these conditions for access in one cloud service.

The basic architecture and flow is shown in Figure 3. A protection API token (PAT) is an OAuth token with a scope that gives the resource server access to the UMA-standardized protection API at the authorization server; an authorization API token (AAT) is an OAuth token with a scope that gives the client access to the UMA-standardized authorization API; and a requesting party token (RPT) is the main access token issued to a requesting party, which does not rely on resource owner presence for issuance.

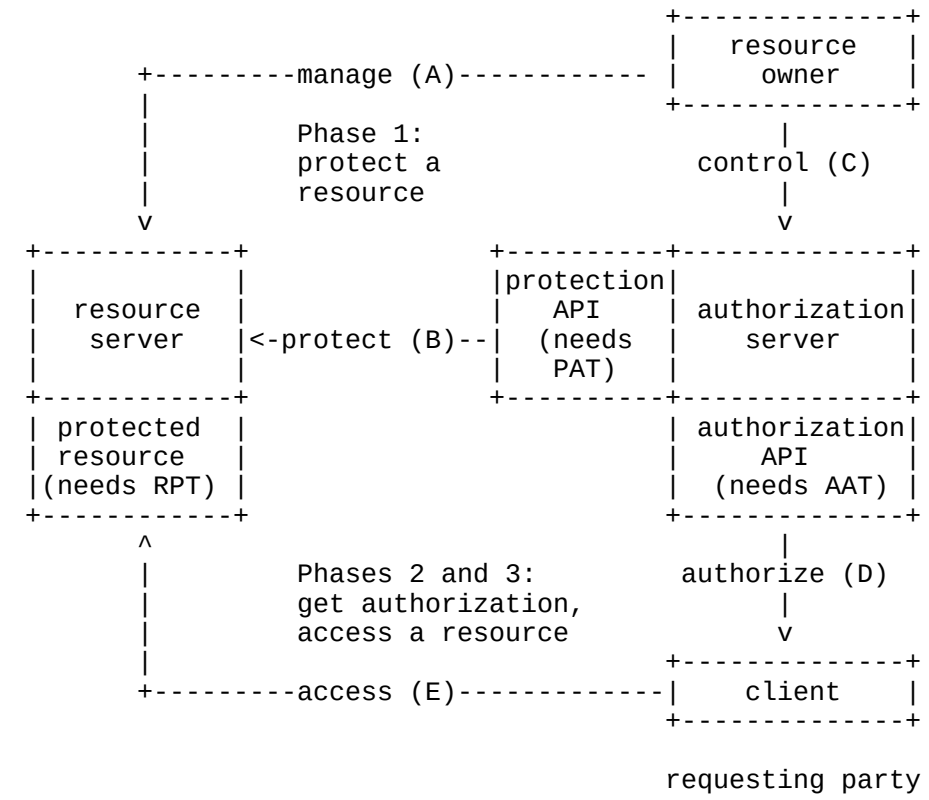


Figure 3: OAuth++: The UMA Architecture.

UMA can be thought of as "OAuth++", in that it adds two major elements: a formal protection API presented by the authorization server, so that resource servers running in different domains can be "authorization relying parties" to it, and the "requesting party" concept distinct from the resource owner (as discussed in Section 2).

The requesting party may be required to interact with the authorization server when the client asks for permission to access a resource. However, if this interaction requires authentication, this authentication step may be outsourced to a variety of different identity providers, including the client (which may be allowed to "push" identity claims to the authorization server), the authorization server itself, or any other identity provider, with the authorization server functioning as a relying party in this case.

Similarly to the previous use case in Section 3.1, there is value in extending the Web world to the world of devices because the data

originating in a device often travels to the cloud. Alice may want to share her scale data with friends, with her doctor, or in anonymized form with a public health service.

The benefit of using an UMA authorization server, requesting party tokens, and so on to manage Alice's control of her doctor's and others' access to the data her scale generates is that she:

1. does not have to be present when they request access, crafting policies prior to access attempts or handling access approval requests after attempts;

2. can demand that requesting parties present proof of their suitability (such as current valid hospital credentials);

3. can change the length permission validity, including revoking sharing relationships;

4. can set policies governing clients used by requesting parties as well; and

5. can do this from a centralizable authorization point, crossing multiple resource servers (and thus devices feeding into them).

3.3. Using OAuth and UMA with Cars

A connected car example illustrates other desirable aspects of IoT authentication and authorization.

Alice buys a new car. At manufacture time, the car was registered at the manufacturer's authorization server. When buying the car, Alice can create an account at the manufacturer's website and reuse the already configured authorization server. Alice installs a car managing mobile app on her phone to manage her car. Alice authorizes the app to act on her behalf as OAuth client to perform actions, such as open car door, which would be similar to authorizing an app to send tweets on my behalf to the twitter API but in this case the resource server is the car and the API is accessed over Bluetooth Smart.

Since the operation of opening the car is security sensitive, it is desirable to require more than a long term access token to open the door and to start the car. So instead of just accepting the access token the authorization server may require Alice to supply more information and a UMA claims gathering process is started, such as requiring a multi-factor authentication using a fingerprint or a PIN code on her phone.

Furthermore, Alice wants to share driving rights with her husband Ted. Alice is owner of the car and is authorized to add new drivers to the car. To do this Alice can setup the policies at the authorization service governing who can do what with the car at what time. Alice configures a rule that allows Ted to request a token for the scope of driving the car, but just as Alice, Ted is required to download the app, authorize it and go through a claims gathering flow to actually get the token to start the car using his smart phone app.

With this delegation of rights to the car Ted could potentially even create a valet key with geo fenced driving range and no access to trunk when he leaves the car in a parking garage and thereby create a valet key for the physical world.

The use of standardized protocols allows Alice to use her own authorization server. Alice could choose to unregister the car at the manufacturer authorization server and register the car to an authorization server of her liking. The car would register available resources and scopes and Alice could configure policies as above using her own authorization server.

Since cars are not always located in areas with Internet connectivity it is envisioned that cars need to be able to verify access tokens locally (without the need to consult an authorization server in real-time). Once the car is online again it could check whether any new revocation information is available and upload information about earlier authorization decisions to the audit log.

A similiar situation may occur when Alice asks her friend Trudy to get the groceries from the trunk of her car (which she forgot there earlier) while they are at their remote summer cottage. Without Internet connectivity Alice cannot delegate access to her car to Trudy using the authorization server located in the cloud. Instead, she transfers an access token to Trudy using Bluetooth. This access token entitles Trudy to open the trunk but not to drive it and grants those permissions only for a limited period. To ensure that the car can actually verify the content of the access token the client app of Alice again uses the capabilities of the proof-of-possession tokens.

3.4. Using OAuth and UMA with Door Locks

Alice, the owner of a small enterprise, buys a door lock system for her office. She would expect to be able to provision policies for access herself, in effect acting as "system administrator" for herself and for her five employees. She may also want to choose her own authorization server, since she wants to integrate the physical access control system with the rest of the resources in her company and the enterprise identity management system she already owns. She

wants to control the cloud-based file system, financial and health data, as well as the version control and issue tracking software.

4. Protocol Designs for the Web and Beyond

The design of OAuth was intentionally kept flexible to accommodate different deployment situations. For example, authentication of the resource owner to the authorization server before granting access is not standardized and different authentication technologies can be used for that purpose. The user interface shown to the resource owner when asking for access to the protected resource is not standardized either.

Over the years various extensions have been standardized to the core OAuth protocol to reduce the need for proprietary extensions that offer token revocation, an access token format called JSON Web Token, or proof-of-possession tokens that offer an alternative security model for bearer tokens [RFC6750].

Due to the nature of the Web, OAuth protocol interactions have used HTTPS as a transport; however, other transports have been investigated as well, such as OAuth for use over SASL (for use with email) and more recently OAuth over the Constrained Application Protocol (CoAP).

This document provides the reader with information about which OAuth extensions will be useful for the IoT context. In its structure it is very similar to the DTLS/TLS IoT profile document that explains what TLS extensions and ciphersuites to use for different IoT deployment environments. Interestingly, very little standardization effort is necessary to make OAuth and UMA fit for IoT. To a large extend the work is centered around using alternative transports (such as CoAP and DTLS instead of HTTP over TLS) to minimize the on-the-wire overhead and to lower code-size and to define profiles for highly demanded use cases.

The UMA group, benefiting from observing the OAuth experience and from the era in which UMA itself has been developed, has built extension points into the protocol, already anticipating a need for flexibility in transport bindings. Thus, UMA has three "extensibility profiles" that enable alternate bindings (such as CoAP) to be defined for communications between an authorization server and resource server, a resource server and client, and an authorization server and client respectively. It also, similarly to OAuth, as other extensibility options, such as token profiling and the ability to extend JSON formats to suit a variety of deployment needs.

5. Instantiations

In this section we provide additional details about the use of OAuth and UMA for solving the use cases outlined in Section 3. In general, the following specifications are utilized:

- OAuth 2.0 [RFC6749] for interacting with the authorization server. The use of the CoAP-OAuth profile [I-D.tschofenig-ace-oauth-iot] maybe used but is not essential for the examples in this section since the client is less constrained.

- Bearer tokens and proof-of-possession tokens as two different security models for obtaining and presenting access tokens. Bearer tokens are defined in [RFC6750] and the architecture for proof-of-possession (PoP) tokens can be found at [I-D.ietf-oauth-pop-architecture]. PoP tokens introduce the ability to bind credentials, such as an ephemeral public key, to the access token.

- UMA [I-D.hardjono-oauth-umacore] for registering the resource server with the authorization server provided by Alice and for management of policy.

- Dynamic Client Registration [I-D.ietf-oauth-dyn-reg] for the client app to register at the authorization server.

- Token introspection [I-D.ietf-oauth-introspection] for optionally allowing the resource server to verify the validity of the access token (if this step is not done locally at the resource server). The use of token introspection over CoAP [I-D.wahlstroem-ace-oauth-introspection] reduces overhead.

- JSON Web Token (JWT) [I-D.ietf-oauth-json-web-token] for the format of the access token. JSON Web Signatures (JWT) [I-D.ietf-jose-json-web-signature] are used for creating a signature over the JWT. The use of a CBOR encoding of various JSON-based security specifications is under discussion to reduce the size of JSON-based tokens.

- A new Bluetooth Smart service and profile for conveying access tokens securely from the client to the resource server. If CoAP runs between the client and a constrained resouce server then [I-D.tschofenig-ace-oauth-bt] provides additional overhead reduction.

5.1. Car Use Case

In the car use case, as described in Section 3.3, the car acts as the resource server and an application on the smart phone plays the role of the client. Alice is first a delegated administrator then becomes a resource owner of the car.

Alice creates an account, downloads and authorizes the mobile app:

1. Alice creates an account on manufacturer’s website.
2. Alice selects that two factor authentication must be used to be able to start controlling car from an app.
3. Alice downloads app and starts it.
4. App has never been provisioned so a browser is started, user selects manufacturer’s authorization server from a list.
5. Alice authenticates using two factors and authorizes the application.
6. Access and refresh tokens are provisioned to the app.

Alice configures policies to add Tim as new driver:

1. Alice opens the car-settings page within the app.
2. Alice selects to add a new driver by supplying Tims email address.
3. Alice checks the checkboxes that also makes Tim a delegated administrator.
4. Alice saves the new policies.

Alice opens car door over Bluetooth Smart:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants to open the car door.
2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the car Alice is seeking access to.
3. The authorization server evaluates the request to open the car door on the specific car and verifies it against the access

control policy. Note that the app authenticated itself to the authorization server.

4. The authorization server prompts Alice for a PIN code using claims gathering.

5. Alice enters pin and the application communicates it to the authorization server.

6. It turns out that the system administrator has granted her access to that specific car and she is given access by returning an access token.

7. The smart phone app then uses the obtained access token to create a request (which includes the access token) over Bluetooth Smart using on the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

8. The car receives the request and verifies it.

9. To check whether the permissions are still valid the car sends the access token to the introspection endpoint.

10. The authorization server validates the access token and returns information about the validity of the token to the car. In this case it’s a valid token.

11. The request is logged.

12. The car gets a response and opens the car door.

Alice changes authorization server:

1. Alice wants to connect the car to her own authorization server instead of the manufacturers default authorization server.

2. Alice makes a request to the current authorization server to unbind the device from the authorization server.

3. The authorization server validates Alice request to remove the authorization server.

4. Alice configures a new authorization server in the apps UI.

5. The app starts an authorization code grant flow with the private authorization server of Alice. Alice logs on and authorizes the app to act on her behalf.

6. The app sends information about the new authorization server to the car using Bluetooth Smart.

7. The car registers the resource it offers with the new authorization server.

8. Alice configures herself as the car owner in the new authorization server.

9. The car unbinds itself from the old authorization server by invalidating the access tokens using the revocation endpoint.

5.2. Door Lock Use Case

In the constrained server use case, as described in Section 3.4, the door lock acts as the resource server and an application on the smart phone plays the role of the client.

Since the client runs on a powerful smartphone standard OAuth according to OAuth Core can be used. To avoid leakage of the access token the use of a proof-of-possession token is utilized instead of a bearer token. This allows the client to demonstrate the possession of the private key to the client. Both symmetric as well as asymmetric cryptography can be used. The use of asymmetric cryptography is beneficial since it allows the client to create a public / private key pair and to never expose the private key to other parties.

As a setup-step the following steps are taken as part of the enterprise IT

1. Alice, as the enterprise network administrator and compay owner, enables the physical access control rights at the identity management server.

2. Alice downloads the enterprise physical access control system app on her phone. By downloading the app she agrees to the terms of use and she accepts the permissions being asked for by the app.

3. Alice associates her smart phone app with her account by login into the enterprise management software, which uses OAuth 2.0 for delegating access to the app.

4. Alice, as the enterprise administrator, configures policies at the authorization server to give her employees access to the office building as well.

5. In this use case each door lock is provisioned with an asymmetric key pair and the public key of the authorization server. The public key of each door lock is registered with the authorization server. Door locks use these keys when interacting with the authorization server (for authentication in case of token introspection), for authenticating towards the client, and for verifying the signature computed over the access token.

When Alice uses her smartphone for the first time to access the office building the following steps take place:

1. The smartphone detects the advertising packets of the door lock and asks Alice whether she wants access.

2. Alice confirms and a request is sent to the authorization server together with an ephemeral public key created by the phone. The request indicates information about the door Alice is seeking access to. The request is protected using TLS.

3. The authorization server evaluates the request and verifies it against the access control policy. Since Alice has added herself to access control policies already she is given access by returning an access token. This access token includes the fingerprint of the public key provided in the request. The access token is digitally signed to avoid any modification of the content.

4. The smart phone app then uses the obtained information to create a request (which includes the access token) over Bluetooth Smart using the (not yet existing) Physical Access Control Profile, which is a security protocol that utilizes public key cryptography where the app demonstrates that it knows the private key corresponding to the finger of the public key found in the token.

5. The door lock software receives the request and verifies the digital signature, inspects the content (such as expiry date, and scope), and determines whether the fingerprint of the public key corresponds to the private key used by the client. Once successfully verified the door is unlocked, and Alice is allowed to enter.

6. The physical access control app caches the access token for future use.

As a variation of the above-described procedure, the door lock might consult the authorization server using token introspection to determine the validity of the access token. This allows the enterprise system software to make real-time access control decisions and to better gain visibility about the number of employees in the building (in case of an emergency).

When Alice approaches the door next time her physical access control app determines that a cached (and still valid) access token is available and no further interaction with the authorization server is needed. Decisions about how long to cache access tokens are a policy decision configurable into the system and impact the performance of the protocol execution.

When Bob, who is employed by Alice, approaches the office building for the first time his downloaded physical access control app also interacts with the door. While Bob still has to consent to the use of app, Alice does not need to authorize access of Bob to the office building in real-time since she has already granted access to her employees earlier already.

6. UMA Use Case Mapping Exercise

An analysis of [I-D.hardjono-oauth-umacore] suggests that its capabilities have a good architectural match with many published ACE use cases. The following are aggregated and paraphrased versions of use cases discussed in [I-D.ietf-ace-usecases]:

Owner grants different resource access rights to different parties (U1.1, U2.3, U.3.2):

> UMA meets this use case because the requesting party is formally distinct from the resource owner and because each requesting party, and each client, is represented distinctly at each authorization server, able to have differential policy applied to it.

Owner grants different access rights for different resources on a device (U1.3, U4.4, U5.2):

> UMA meets this use case because the resource server is able to register each resource set (according to boundaries it unilaterally determines) at the authorization server, so that the resource owner can apply policy to it distinctly.

Owner not always present at time of access (U1.6, U5.5):

UMA meets this use case because it is a profile of OAuth that defines an asynchronous authorization grant, meaning that the client's interactions during a resource access attempt do not require a resource owner's interaction.

Owner grants temporary access permissions to a party (U1.7):

UMA meets this use case because the default, mandatory-to-implement permissions associated with a requesting party token (the "bearer" profile) are able to be time-limited and are in a time-limitable JSON Web Token as well.

Owner applies verifiable context-based conditions to authorizations (U2.4, U4.5, U6.3):

UMA meets this use case because a resource owner can configure an authorization server with policies, or an authorization server can apply system-default policies, to demand "trust elevation" when a client requests authorization data, such that a requesting party or client must satisfy authentication, claims-based, or (through extension) any other criteria prior to being issued authorization data.

Owner preconfigures access rights to specific data (U3.1, U6.3):

UMA meets this use case because it defines an asynchronous authorization grant, as described above. Preconfiguration is a case when a resource owner sets policy prior to an access attempt.

Owner adds a new device under protection (U4.1):

UMA meets this use case because it enables a resource owner to associate a device and its corresponding resource server with an authorization server through consenting to the issuance of a protection API token (PAT), enabling the resource server to outsource protection of its resources to the authorization server.

Owner puts a previously owned device under protection (U4.2):

UMA meets this use case because a previous resource owner can revoke a pre-existing PAT if one existed, revoking the previous consent in place, and the new owner can mint a new PAT.

Owner removes a device from protection (U4.3):

UMA meets this use case because the resource owner can revoke the PAT.

Owner revokes permissions (U4.6):

UMA meets this use case because the resource owner can configure the authorization server to revoke or terminate an existing permission. The default, mandatory-to-implement requesting party token profile ("bearer") requires runtime token introspection, ensuring relatively timely retrieval of a revoked permission (barring authorization server caching policy). Other profiles may have different results.

Owner grants access only to authentic, authorized clients (U7.1, U7.2):

UMA meets this use case because it enables OAuth as well as OpenID Connect authentication of clients, including dynamic authentication, and also enables resource owners to configure authorization servers with policy, such that only desired clients wielded by desired requesting parties are given access to the owner’s resources.

7. Security Considerations

This specification re-uses several existing specifications, including OAuth and UMA, and hence the security-related discussion in those documents is applicable to this specification. A reader is encouraged to consult [RFC6819] for a discussion of security threats in OAuth and ways to mitigate them. On a high level, the security guidance provided in [I-D.iab-smart-object-architecture] will help to improve security of Internet of Things devices in general.

Despite all the available guidance it is nevertheless worthwhile to repeat the most important aspects regarding the use of access tokens, which are a core security mechanism in the OAuth / UMA specifications.

Safeguard bearer tokens: Client implementations MUST ensure that bearer tokens are not leaked to unintended parties, as they will be able to use them to gain access to protected resources. This is the primary security consideration when using bearer tokens and underlies all the more specific recommendations that follow. This document also outlines the use of proof-of-possessions, which provide stronger security properties than bearer tokens and their use is RECOMMENDED.

Validate TLS certificates: TLS/DTLS clients MUST validate the certificates received during the handshaking procedure. TLS/DTLS is used heavily in OAuth/UMA between various parties. Failure to verify certificates will enable man-in-the-middle attacks.

Always use TLS/DTLS: The use of TLS/DTLS is mandatory for use with OAuth as a default. Particularly when bearer tokens are exchanged the communication interaction MUST experience communication security protectoin using TLS (or DTLS). Failing to do so exposes bearer tokens to third parties and could consequently give attackers unintended access. Proof-of-possession tokens on the other hand do not necessarily require the use of TLS/DTLS but TLS/DTLS is RECOMMENDED even in those cases since TLS/DTLS offers many desireable security properties, such as authentication of the server side.

Issue short-lived tokens: Authorization servers SHOULD issue short-lived tokens. Using short-lived bearer tokens reduces the impact of them being leaked and allows easier revocation in scenarios where resource servers are offline.

Issue scoped tokens: Authorization servers MUST issue tokens that restrict tokens for use with a specific resource server and contains appropriate entitlements to control access in a fine-grained fashion.

8. IANA Considerations

This document does not require actions by IANA.

9. Acknowledgements

This is the first version of the document. We appreciate feedback.

10. References

10.1. Normative References

[I-D.hardjono-oauth-umacore]
Hardjono, T., Maler, E., Machulak, M., and D. Catalano, "User-Managed Access (UMA) Profile of OAuth 2.0", draft-hardjono-oauth-umacore-12 (work in progress), February 2015.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 2015.

[I-D.ietf-oauth-dyn-reg]
ietf@justin.richer.org, i., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", draft-ietf-oauth-dyn-reg-24 (work in progress), February 2015.

[I-D.ietf-oauth-introspection]
ietf@justin.richer.org, i., "OAuth 2.0 Token Introspection", draft-ietf-oauth-introspection-05 (work in progress), February 2015.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-32 (work in progress), December 2014.

[I-D.ietf-oauth-pop-architecture]
Hunt, P., ietf@justin.richer.org, i., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-bt]
Tschofenig, H., "The OAuth 2.0 Bearer Token Usage over the Constrained Application Protocol (CoAP)", draft-tschofenig-ace-oauth-bt-01 (work in progress), March 2015.

[I-D.tschofenig-ace-oauth-iot]
Tschofenig, H., "The OAuth 2.0 Internet of Things (IoT) Client Credentials Grant", draft-tschofenig-ace-oauth-iot-01 (work in progress), March 2015.

[I-D.wahlstroem-ace-oauth-introspection]
Wahlstroem, E., "OAuth 2.0 Introspection over the Constrained Application Protocol (CoAP)", draft-wahlstroem-ace-oauth-introspection-00 (work in progress), October 2014.

[OIDC] Sakimura, N., "OpenID Connect Core 1.0 incorporating Errata Set 1", http://openid.net/specs/openid-connect-core-1_0.html, November 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

[RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

10.2. Informative References

[I-D.iab-smart-object-architecture] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", draft-iab-smart-object-architecture-06 (work in progress), October 2014.

[I-D.ietf-ace-usecases] Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-02 (work in progress), February 2015.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

Authors' Addresses

Hannes Tschofenig
ARM Limited
Austria

Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Eve Maler
Forgerock

Email: eve.maler@forgerock.com

Erik Wahlstroem
Nexus Technology
Sweden

Email: erik.wahlstrom@nexusgroup.com
URI: https://www.nexusgroup.com

Samuel Erdtman
Nexus Technology
Sweden

Email: samuel.erdtman@nexusgroup.com
URI: https://www.nexusgroup.com

ACE WG
Internet-Draft
Intended status: Informational
Expires: January 29, 2016

A. Rahman
C. Wang
V. Choyi
InterDigital Communications, LLC
July 28, 2015

Public Safety Use Case
draft-rahman-ace-public-safety-use-case-01

Abstract

A public safety use case is proposed for consideration by the ACE WG.

Table of Contents

1. Terminology and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This document assumes readers are familiar with the terms and concepts that are used in [RFC7252] and [I-D.ietf-ace-usecases].

2. Background

The ACE WG, as per their charter, is in the process of defining use cases to drive the specification of standardized solutions for authentication and authorization to enable authorized access (Get, Put, Post, Delete) to resources identified by a URI and hosted on a resource server in constrained environments. As a starting point, the WG will assume that access to resources at a resource server by a client device takes place using CoAP and is protected by DTLS ([RFC7252]). Both resource server and client may be constrained. This access will be mediated by an authorization server, which is not considered to be constrained.

3. Public Safety Use Case

A Fire Department requires that as part of the building safety code, that the building have sensors that sense the level of smoke, heat, Etc., when a fire breaks out. These sensors report metrics which are then used by a back-end server to map safe areas and un-safe areas within a building and also possibly the structural integrity of the building before fire-fighters may enter it. Sensors may also be used to track where human/animal activity is within the building. This will allow people stuck within the building to be guided to safer areas and suggest possible actions that they make take (e.g. using a client application on their phones, or loudspeaker directions) in

order to bring them to safety. In certain cases, other organizations such as the Police, Ambulance, and federal organizations are also involved and therefore the co-ordination of tasks between the various entities have to be carried out using efficient messaging and authorization mechanisms.

4. Authorization Problem Summary

1. The principal wants to ensure that only authorized clients can read data from sensors and send commands to the actuators.

2. The principal wants to be able to grant access rights dynamically when needed. This may be triggered where the Principal may be human or machine (server/cloud system).

3. The principal wants to ensure the authenticity of the data originating from the sensors (authenticating the originator of data).

4. The principal wants to ensure the Integrity of the received data.

5. The principal wants to ensure that data sent to the actuators are Integrity protected.

6. The principal wants to ensure that extremely time-sensitive operations have to be carried out in a quick manner.

7. The principal wants to ensure the ability to prove that an entity (e.g. police or fire chief) that issued a message had indeed issued the message (authenticating the sender of the message).

8. The principal wants to ensure that all the messaging and data involved during a crisis is audit-able in a transparent manner.

5. Acknowledgements

TBD.

6. IANA Considerations

This memo includes no request to IANA.

7. Security Considerations

The entire draft is regarding a security use case related to authentication and authorization for constrained environments.

8. References

8.1. Normative References

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-04 (work in progress), June 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

Authors' Addresses

Akbar Rahman
InterDigital Communications, LLC

Email: akbar.rahman@interdigital.com

Chonggang Wang
InterDigital Communications, LLC

Email: chonggang.wang@interdigital.com

Vinod Choyi
InterDigital Communications, LLC

Email: vinod.choyi@interdigital.com

ACE Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2015

L. Seitz
SICS
G. Selander
Ericsson
M. Vucinic
STMicroelectronics
June 29, 2015

Authorization for Constrained RESTful Environments
draft-seitz-ace-core-authz-00

Abstract

This memo defines a framework for authorization in constrained-node networks, i.e. networks where some devices have severe constraints on memory, processing, power and communication bandwidth. The main goal is to offload constrained devices by providing them access control support from trusted parties that are less constrained. The approach is based on RESTful requests to dedicated access management resources, supporting different authorization schemes and communication security paradigms.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 31, 2015.

Table of Contents

1. Introduction

Authorization is the process of deciding what an entity ought to be allowed to do. Authorization management and processing access control policies in order to reach an authorization decision is often a heavyweight task, involving e.g. database lookups, credential verification and matching large sets of values against each other. Constrained devices are ill-equipped to handle this on their own, therefore the logical approach is to offload authorization management and part of the decision process to a less constrained, trusted third party.

This memo describes an approach to authorization in constrained-node networks using dedicated resources for access management. The various entities in the architecture handle authorization information by making RESTful requests (GET/PUT/POST/DELETE) to these resources and thereby establish the necessary security contexts for interactions between endpoints.

The approach is based on the use cases from [I-D.ietf-ace-usecases] and on the architecture and problem description from [I-D.gerdes-ace-actors]. It introduces protocols for requesting authorization information from a trusted third party, encodings for such authorization information and protocols for transferring it to the affected device(s). We assume that CoAP [RFC7252] is the preferred application-layer protocol and that constrained devices implementing these mechanisms can be as resource constrained as class 1 according to the definitions in [RFC7228].

An important goal in the design of this framework is to preserve end-to-end security in the presence of untrusted intermediate nodes in the communication of requests, responses and related authorization information.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Certain security-related terms are to be understood in the sense defined in [RFC4949]. These terms include, but are not limited to, "authentication", "authorization", "confidentiality", "(data) integrity", "message authentication code", and "verify".

RESTful terms including "resource", "representation", etc. are to be understood as used in HTTP [RFC7231] and CoAP [RFC7252].

Terminology for constrained environments including "constrained device", "constrained-node network", "class 1", etc. is defined in [RFC7228].

Terminology for entities in the architecture is defined in [I-D.gerdes-ace-actors], such as the constrained nodes Client (C) and Resource Server (RS), and the less constrained nodes Client Authorization Server (CAS) and Authorization Server (AS).

Since this draft focuses on the problem of access control to resources it is for simplicity assumed that the Client Authorization Server functionality is not stand-alone but subsumed by either the Authorization Server or the Client (see section 2.2 in [I-D.gerdes-ace-actors]).

We use the term "authorization token" for authorization information that is protected with object security.

2. Overview

This section contains an overview of the authentication and authorization problem and the solution described in this memo. We begin with two generic examples.

2.1. Example A: Access to an Actuator

Consider an actuator, e.g. in an industrial control system. Set in the terms of the architecture [I-D.gerdes-ace-actors], the actuator is hosted in one endpoint, acting RS, and another endpoint, acting C, is requesting access to the actuation resource.

Security requirements of special importance are:

- RS needs to verify that the requesting Client is authorized to access the resource.
- Integrity and replay protection of request. Assuming CoAP, integrity protection applies to Payload but also to other fields such as for example the CoAP option Uri-path.
- C must be able to verify the integrity of the response, and that the received message is the response to the previously made request, in order for C to verify that the actuation was performed.
- It must be possible to confidentiality protect the request and response, e.g. for privacy reasons.

2.2. Example B: Multiple Devices Accessing Sensor Data

Consider a constrained sensor performing measurements consumed by multiple devices. Since the communication of a duty-cycled sensor with each consuming device would require energy expensive radio receptions and transmissions, it is favorable to minimize the number of devices that directly interact with the sensor. Therefore the sensor publishes (e.g. sends notifications with) measurements to a message broker, which forwards the measurements to subscribing devices. Access control is performed by only giving plaintext access to authorized subscribers.

In the terminology of [I-D.gerdes-ace-actors] this means that the sensor acts RS and the subscribing devices are Clients. In this case, the RS may neither be able to verify integrity nor replay of individual client requests, nor authorize individual clients, since it isn't necessarily interacting directly with any clients. For the same reason the client is not able to match a publication against a request. The main relevant security requirements here are:

- Access to plaintext publications must be restricted to authorized Clients.
- A Client needs to verify the origin of the publication, and that it is not a replay of an old publication.

2.3. The Authentication and Authorization Problem

Both examples in the previous subsections are encompassed by the problem statement described in [I-D.gerdes-ace-actors] but their security requirements are quite different. On a high level, the authentication and authorization problem can be broken down into three sub-problems:

1. The policy for resource access as defined by the Resource Owner (RO) and managed by the Authorization Server (AS) needs to be translated into authorization information following an Authorization Information Format (AIF) that the RS can parse and act upon.
2. Similarly, the Client may need to acquire keys/credentials or other information from the AS to securely access the resource. Such client information is encoded following a Client Information Format (CIF).

3. In order for Client and RS to authenticate and authorize according to policy, the communication of the various information elements needs to be protected appropriately end-to-end: from AS to C, from AS to RS, and between C and RS.

To access an actuator, as in example A, the authorization information could be an authorization decision that C is granted access to a certain resource, or attributes of C that RS can match against internal access policies. The client information could be cryptographic keys which C could use to establish secure communication with RS.

In the publish-subscribe scenario of example B, the authorization information (if present) could be cryptographic keys that must be used for protection of published resource representations. The client information could be cryptographic keys/credentials with which publications of certain resource representations can be verified and decrypted.

Communication security could be addressed with a session security based protocol such as DTLS [RFC6347], or an object security solution based on e.g. COSE [I-D.schaad-cose-msg] (see Section 4).

The other sub-problems listed in this section, AIF and CIF, are further detailed in Section 6 and Section 7, respectively.

2.4. Solution Overview

Considering the large variety of use cases, it may be difficult to address the authentication and authorization problem with one single protocol instance. In this section we list some guiding principles for a solution. The remainder of the document provides solution components based on the principles listed in this section. In Section 8 the detailed processing steps are presented. In Appendix A examples of the proposed solution are given.

PRINCIPLE 1: Allow different order of information flows

The information flows, including AIF, CIF, and Resource Request/ Response (RRR) (Figure 1), may take place in different order see Section 3.

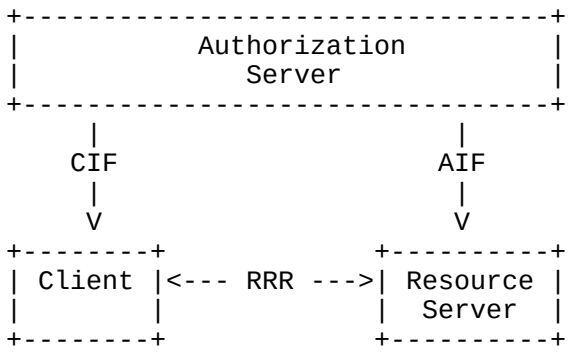


Figure 1: Information flows

PRINCIPLE 2: Build upon REST

The information flows should leverage the RESTful architecture. For RRR this is already assumed, but RESTful administrative resources for managing access to e.g. AIF and CIF need to be defined. The support for REST should not preclude optimizations for particular protocols, for example some flows may be embedded within the DTLS handshake.

PRINCIPLE 3: Allow security at different layers

Leave to the application to decide about session based or object based security for any given information flow.

3. Information Flows

This section gives an overview of information flows between the entities in the architecture, and the use of access management resources.

3.1. Message Sequences

The information flows in scope between the nodes (constrained to constrained and constrained to less constrained) are described in [I-D.gerdes-ace-actors]. Taking [RFC2904] as a starting point, there are three authorization schemes, defining the message sequences for the flow of request, response, and authorization data:

1. In the Push scheme, C first obtains authorization information from AS and then transmits this data to RS together with its request, and finally RS sends its reply. In this model RS does not communicate with AS directly.

2. In the Pull scheme, C sends its request to RS directly, RS then contacts AS in order to obtain authorization information, AS provides authorization information, and finally RS sends back the reply. In this model C does not communicate with AS directly.

3. In the Agent scheme, C sends its request to AS, which evaluates C's access rights and, if authorized, executes the request on C's behalf, transmitting RS's response back to C. In this model C does not communicate with RS directly.

Depending on use case, different schemes may be more appropriate than others. From a constrained-node network point of view we note the following:

- The Push and Agent scheme requires C to connect to AS.

- The Pull and Agent scheme requires RS to connect to AS.

- The Pull scheme requires RS to handle two secure connections at the same time (RS - AS and RS - C) in order to perform the real time authorization check.

Considering processing and memory requirements of RS, the Push scheme is more favorable than the Pull scheme.

The Agent scheme assumes constant connectivity with both C and RS and is therefore not applicable to the constrained setting.

However, the publish-subscribe case from Section 2.2 is not covered by any of the schemes so this memo defines a new scheme, called Client-Pull.

4. In the Client-Pull scheme, RS first provides unconditional access to a protected resource representation. C obtains this protected resource representation and then contacts AS to obtain authorization information (in this case cryptographic keys) for access and verification of the resource representation. In this model RS does not communicate with AS directly, and C does not communicate with RS directly.

The Client-Pull scheme is only useful for GET access. However, it offloads the RS considerably, since the RS neither receives any requests from clients nor performs any access control verifications. Furthermore it enables use cases, such as caching and publish-subscribe, that can not be covered by the other schemes in a satisfactory manner.

3.2. Access Management Resources

This section describes an approach for transferring authorization information between the various entities in the architecture by making RESTful requests to dedicated resources for access management. The structure or naming of such "access management resources" is left for a future version of this memo, and for simplicity we denote any such resource "/authorize".

In this section we illustrate various message sequences for transferring authorization information by means of requesting access management resources. We assume, as in section 3.2 of [I-D.gerdes-ace-dcaf-authorize], that the need for transferring authorization information may be triggered by a request from C to RS to access a resource. Figure 2 shows C making an access request (GET /PUT/POST/DELETE) which cannot be granted by the RS because of missing authorization information.

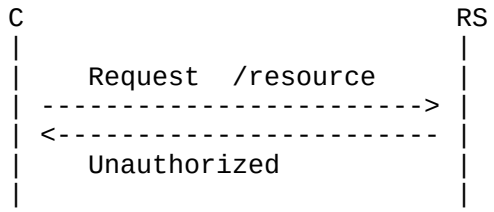


Figure 2: Access request unauthorized.

Starting with the Pull scheme, if the RS can access the AS, then any missing authorization information could be requested on the fly, see Figure 3. The RS sends an authorization request in a specific Authorization Request Format (ARF) to the /authorize resource at the AS as detailed in Section 8 and receives authorization information in a specific Authorization Information Format (AIF) in the response. RS updates the stored authorization information based on the new information and executes the client’s resource request accordingly. Note that C and RS need be mutually authenticated using pre-established credentials in this scheme.

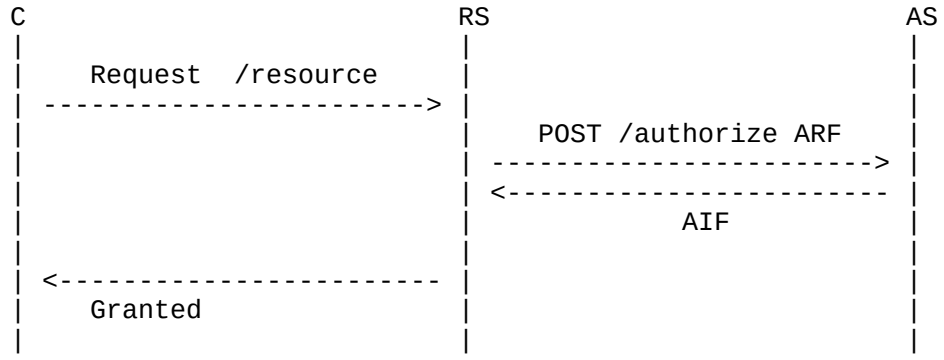
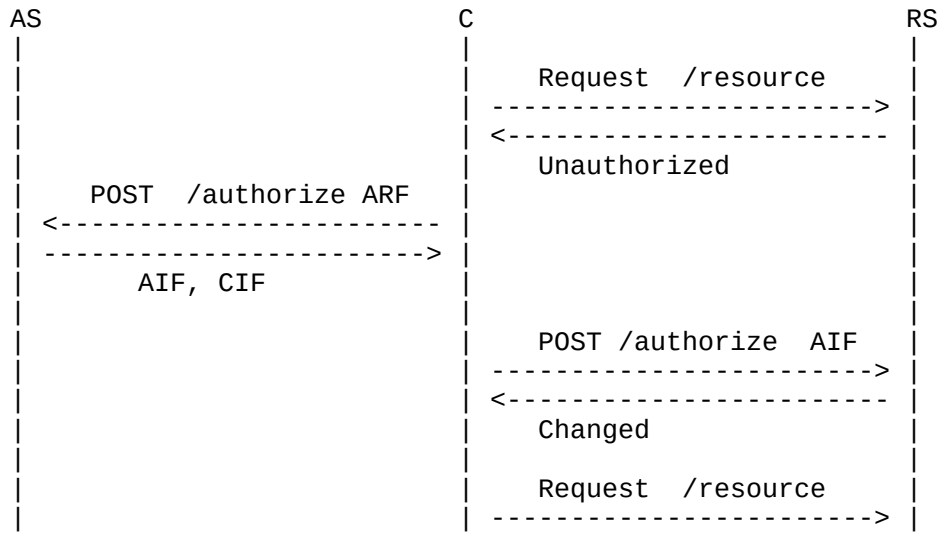


Figure 3: Pull Scheme

In use cases where the RS does not have connectivity with AS, or if the RS cannot handle two secure connections at the same time, the Push scheme provides an alternative, see Figure 4. C receives, in the negative response to its resource request, information about where to request the missing authorization information. In this case it is C that requests the resource /authorize at the AS and receives both authorization information, and client information in a specific Client Information Format (CIF). The latter will be used by C to establish communication security with RS. Next, C pushes the authorization information to the /authorize resource at the RS. RS updates the stored authorization information based on the new data and confirms the change, after which C can repeat the resource request to the RS with the new client information and authorization information in place.



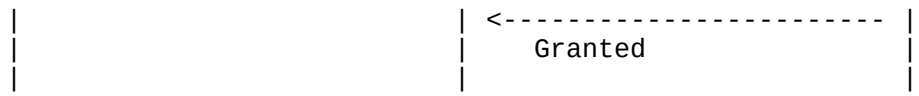


Figure 4: Push Scheme

Note that in case there is connectivity between RS and AS, but RS has not implemented the Pull scheme or cannot handle two secure connections at the same time, then AS could POST the authorization information to the /authorize resource of RS directly instead of using C as intermediary node.

Also note that the authorization information needs to be integrity protected during the transport between AS and RS. This is because the client has access to it and would otherwise be able to change its permissions. How this is done is described in Section 4.

Finally the Client-Pull scheme, which only requires C to request a Protected Resource Representation from the RS, and client information (e.g. cryptographic keys) from the AS.

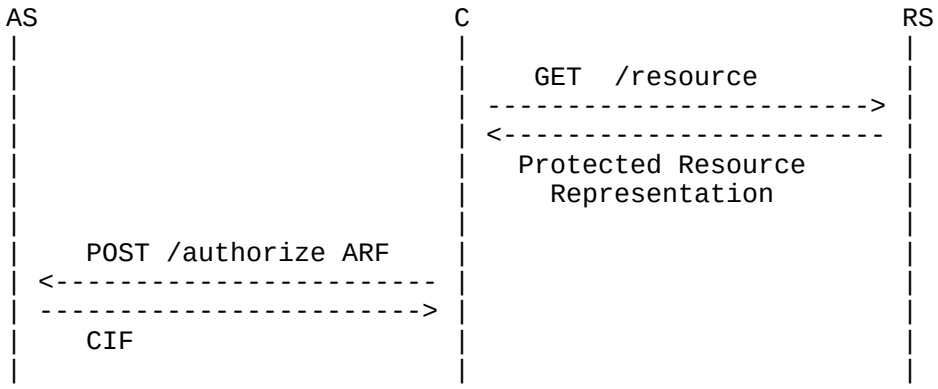


Figure 5: Client-Pull Scheme

Note that the order of the information flows is different for the different authorization schemes. The authorization scheme is defined at design time, so an RS would typically support one specific message sequence suitable for the particular application.

4. Communication Security

In this section we address the third sub-problem of Section 2.3, the authentication and communication security problem, based on the assumptions made in [I-D.gerdes-ace-actors]:

- AS and RS are assumed to have established security contexts (keys, credentials, security protocols, parameters). How this was established is out of scope for this work.

- AS is assumed to have a policy for how C is allowed to access the resources of RS, including information for how to authenticate C, such as a shared secret key, public key or other credential.

- Both session based and object based security solutions shall be supported.

Since CoAP is the default communication protocol, DTLS [RFC6347] is the default session based security protocol. DTLS MUST NOT be used with untrusted intermediary nodes. For object security in constrained environments, COSE [I-D.schaad-cose-msg] is the main candidate. OSCOAP [I-D.selander-ace-object-security] defines a profile of COSE, suitable for securing individual CoAP messages, and two Modes of operation:

- Mode:PAYL for (CoAP) Payload confidentiality, integrity and replay protection.

- Mode:COAP for additional protection of CoAP Headers (Code) and Options (Uri-Path, Uri-Query etc.) and for securing request-response pair.

4.1. AS - RS

The main purpose of this communication is to provide authorization information from AS to RS. There are three communication patterns:

1. RS: POST /authorize to AS (ARF in request, AIF in response)

2. AS: POST /authorize to RS (AIF in request)

3. C: POST /authorize to AS (ARF in request, AIF in response) followed by C: POST /authorize to RS (AIF in request)

Pattern 1. and 2. SHALL be protected by DTLS or COSE Mode:COAP, leveraging the pre-established security context. In pattern 3., since C is an untrusted intermediary node, the authorization information SHALL be protected end-to-end from AS to RS with COSE Mode:PAYL, leveraging the pre-established credentials.

Pattern 2. with COSE Mode:COAP includes the case of POST via an intermediary untrusted Forward Proxy, which in particular could be the Client, if the Client supports Forward Proxy functionality.

In pattern 3. the use of transport layer security (DTLS) or CoAP message security (Mode:COAP) is optional, since Mode:PAYL protected authorization information is in itself a confidential, integrity and replay protected token.

If the subsequent communication between C and RS is going to be protected by DTLS, then an OPTIONAL optimization of the message exchange is to replace "C: POST /authorize to RS (AIF in request)" with a transfer of the protected authorization in the DTLS Handshake, see Section 4.4.

4.2. AS - C

The main purpose of this communication is to provide client information from the AS to the C. The communication pattern is:

- C: POST /authorize to AS to (ARF in request, CIF in response)

This pattern SHALL be protected by DTLS or COSE Mode:COAP, leveraging out of band established credentials.

4.3. C - RS

This is the original resource request/response problem. Since there may be no pre-established keys between C and RS, it is the purpose of AIF and CIF to provide information for authentication and communication security. There are two communication patterns:

1. The Client makes a request (GET/PUT/POST/DELETE /resource) to RS and receives a response.

2. The Client makes a GET request to the RS, or a cache, or message broker and unconditionally receives a protected resource representation

Pattern 1. SHALL be protected by DTLS or COSE Mode:COAP, leveraging the established keys or credentials in authorization information and client information. Pattern 2. SHALL be protected with COSE Mode:PAYL.

4.4. Authorization Information transfer in the DTLS Handshake

In certain situations, a session security protocol like DTLS [RFC6347] can be used directly to send the authorization information to the RS and optimize the message exchange. The authorization information MAY be embedded in the DTLS handshake. In this case the authorization information SHOULD be transferred using the TLS supplemental data extension [RFC4680].

Figure 6 illustrates this approach for a DTLS handshake. The messages marked with * are optional depending on the type of handshake.

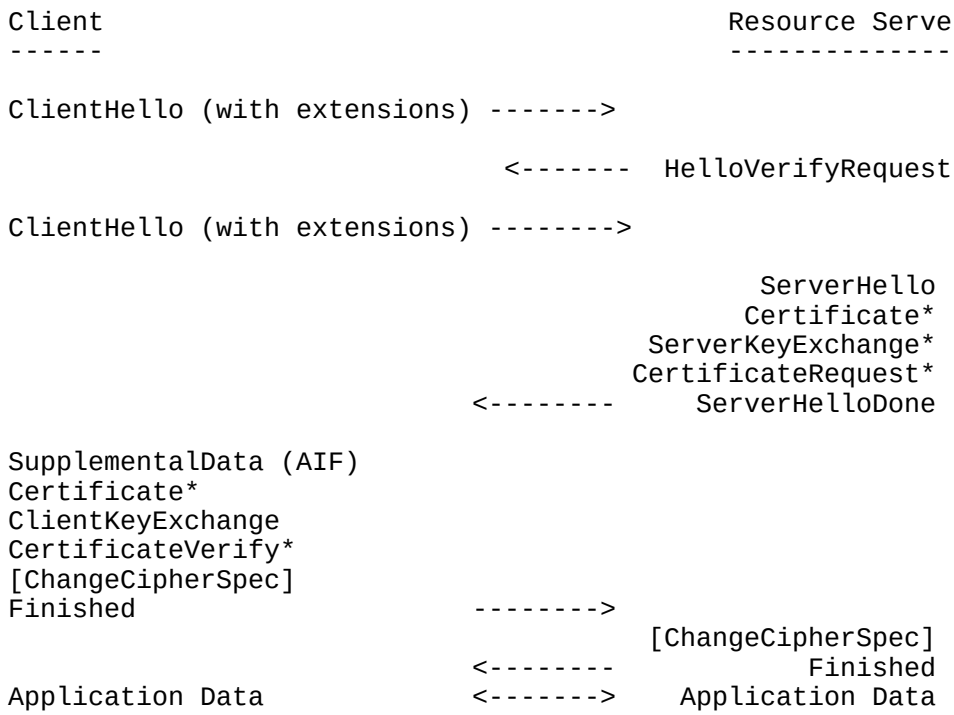


Figure 6: Authorization information Transfer in Handshake

The SupplementalDataType value SHOULD be set to 16386 (authz_data). The Resource Server MUST verify the validity of the received authorization information before accepting any application data from the client.

The RS MUST verify that the authorization information transmitted in the SupplementalData message is bound to the same key as the one the Client used in the handshake (see Section 6 for the subject binding in the AIF). Note that this also enables AS-asserted authorization of the DTLS handshake.

5. Authorization Request Format

Authorization requests are represented in the Authorization Request Format (ARF). They are produced either by the Client (Push, Client-Pull) or the RS (Pull) and consumed by the AS. The ARF allows to specify requests for several actions on several resources in a single message.

5.1. ARF Information Model

The general information that needs to be contained in the ARF is the following:

- The identifier of the subject of this authorization request (usually the Client).
- The identifier of the host that provides the resources that are requested in this authorization request (CoAP option URI-host). Note that the ARF does not allow to specify more than one host.
- The requested resources and actions on these resources.

The resources can be represented by the "path-absolute" part of the URI (In CoAP this would be the Uri-Path options).

5.2. ARF Data Model

For representing the ARF discussed in Section 5.1 the following steps are performed:

- The subject identifier is encoded in a map that maps the type of the identifier to its value. The type could e.g. be a raw public key, or the key identifier of a secret key, or a X.509 distinguished name.
- The host identifier is encoded as second parameter.
- The third parameter is array of resource-actions tuples. The structure of the entries is specified as follows:
  - Requests that affect the same resource are merged into a single entry that specifies the union of the permitted actions.
  - The actions GET, POST, PUT, DELETE are represented as integer 0, 1, 2, and 3 respectively.

* The set of numbers is converted into a single number by taking each number to the power of two and computing the inclusive OR of the binary representations of all the numbers.

* Each entry is an array, containing the resource identifier, and the number representing the actions.

This information can then be represented in CBOR [RFC7049] or JSON [RFC7159] as an array of the elements listed above. An example JSON representation of the ARF is given in Figure 7.

```
[{"subjectKeyId":"someKeyId1"}, "rs.example.com",
 [["sensors/tempC",1], ["conf/sleepInterval", 5],
  ["conf/sleepDuration", 5]]]
```

Figure 7: Example JSON representation of an access request

6. Authorization Information Format

Addressing the first sub-problem of Section 2.3, this section defines an Authorization Information Format (AIF) for encoding authorization information. AIF instances are either consumed directly by an RS (Pull), or packaged into a COSE Mode:PAYL token and sent to the Client, who forwards the token to the RS (Push).

This memo defines two different types of AIF that can be used by an AS:

- One or more access control decisions, listing one or more tuples of actions and resources similar to a capability list. This is a simple super-set of the ARF.

- Group memberships for the Client. This type of AIF is used in a configuration where the RS stores access control lists (ACLs) corresponding to a number of groups. The RS will resolve a specific group to a specific set of ACLs which are then applied to the request.

6.1. AIF Information Model

The general information that needs to be contained in the AIF is the following:

- The identifier of the subject of this authorization information (usually the Client)

- The identifier of the host that provides the resources for which this authorization information applies (CoAP option URI-host). Note that the AIF does not allow to specify more than one host.

If the authorization information represents one or more group memberships, the only other information needed is the group names. If the authorization information represents access control decisions, the affected resources and the actions are needed.

Authorization decisions can also MAY contain local conditions, that can only be evaluated by the RS at access time (e.g. "NotBefore='09:00', NotAfter='18:00'"). This memo does not define the format of these local conditions, as it is expected that these will be application specific. The RS MUST reject any authorization information that contains local conditions that it does not understand.

6.2. AIF Data Model

For representing the AIF discussed in Section 6.1 the following steps are performed:

- The subject identifier is encoded in a map that maps the type of the identifier to its value. The type could e.g. be a raw public key, or the key identifier of a secret key, or a X.509 distinguished name. Section 4.3 defines how this is used to bind the authorization information to a specific client.

- The host identifier is encoded as second parameter.

- The third parameter is either an array of access control decisions, or if group memberships are asserted, a map, mapping the key 'grp' to the array of group names. The structure of the entries in the access control decision array is the same as for the ARF, except that is may also contain local conditions represented as strings.

- Each entry is an array, containing the resource identifier, the number representing the actions, and optionally the local conditions.

This information can then be represented in CBOR [RFC7049] or JSON [RFC7159] as an array of the elements listed above. An example JSON representation of the AIF with access control decisions is given in Figure 8, while an example of the AIF containing group membership assertions is shown in Figure 9.

```
[{"SubjectKeyId":"someKeyId2"}, "rs.example.com",
 [["actuators/doorLock", 5, "not-before:’07:00’;not-after:’18:00’"]]]
```

Figure 8: Example JSON representation of access control decisions

```
[{"SubjectPublicKey":"MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEJG4m93
ED8tPK2CpkhrrtKNx1vXDbml4Fun1628Qkl1U_aIB5zUfqPwaacznbqoMJ6vQVZ7
4X9HpfynouLK_ujw"}, "rs.example.com",
{"grp":["admin","user"]}]
```

Figure 9: Example JSON representation of group memberships

Note that different subject identifiers might be used in instances of the ARF and the AIF in the same Push scheme. This is due to the fact that the credential the Client uses towards the AS may not be the same as the one later used towards the RS.

7. Client Information Format

This section addresses the second sub-problem of Section 2.3, and describes a format for client information. Two types of client information are defined in this section:

1. Client Information related to the Push scheme, which encodes keys and related parameters, that allow the Client to communicate securely with the RS and prove that it is the legitimate subject of some authorization information.

2. Client Information related to the Client-Pull scheme represent keys and related parameters that allow the Client to access and verify a resource representation protected with object security.

For the Push scheme, the CIF must be able to encode instructions to the Client on how secure the connection between Client and RS, and how to transfer the authorization information to the RS.

7.1. CIF for the Push scheme

The encoding of the client information in a Push scheme is an array with the following elements:

1. The type of this client information, here the string ’push’.

2. The method to be used for securing the transfer. Either the string ’coaps’ (for CoAP over DTLS) or the string ’oscoap’ (for CoAP with object security).

3. The method of transferring the authorization information. Either the string 'suppl' (for supplemental data) or the URI of the resource on the RS to which the authorization information should be transferred.

4. The type of key to be used for coaps of oscoap. This can either be a raw public key denoted by the string 'rpk', or a pre-shared key denoted by the string 'psk'.

5. If the key type was 'psk', then the next element is the identifier of this key. If the key type was 'rpk' this element is the raw public key of the RS. The encoding here should follow the SubjectPublicKeyInfo defined in RFC 7250 [RFC7250].

6. If the key type was 'psk', then the next element is the pre-shared key that the Client should use either for DTLS or object security. The encoding of this key depends on the representation format. For CBOR it is raw bytes, for JSON it is Base64 encoded bytes.

Figure Figure 10 shows an example representation of the CIF for a Push scheme, using JSON.

```
["push", "oscoap", "/authorize", "psk",
"someKeyId", "Z4LXBNbOeeOJOYglBLb4pg"]
```

Figure 10: CIF example for the Push scheme

7.2. CIF for the Client-Pull scheme

The encoding of the CIF for a Client-Pull scheme is an array with the following elements:

1. The type of this client information, here the string 'cpull'.

2. The public key of the RS used to sign the resource representation. The encoding here should follow the SubjectPublicKeyInfo defined in RFC 7250 [RFC7250].

3. The secret key used to encrypt the resource representation. The encoding of this key depends on the representation format. For CBOR it is raw bytes, for JSON it is Base64 encoded bytes.

Note that further parameters are provided in the secure object itself, such as e.g. algorithm identifiers and initialization vectors.

Figure Figure 11 shows an example representation of the CIF for a Client-Pull scheme, using JSON.

```
["cpull", "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEJG4m93
ED8tPK2CpkhrrtKNx1vXDbml4Fun1628Qkl1U_aIB5zUfqPwaaczn
bqoMJ6vQVZ74X9HpfynouLK_ujw", "eyJhbGciOiJIUzI1NiIsIm"]
```

Figure 11: CIF example for the Client-Pull scheme

8. Message Processing

This section puts together the pieces from previous sections and specifies more in detail the processing steps for the authorization schemes defined in Section 3.

8.1. Unauthorized Access Attempt

All schemes can start with the Client requesting access at the RS without any established authorization information. Depending on which authorization scheme (Push, Pull, Client Pull) the RS has implemented, the RS can either:

- Query the AS using the Pull scheme.
- Instruct the Client to use the Push scheme.
- In case the Client-Pull scheme, provide an encrypted representation of the resource that was requested, without performing any access control.

If the RS wants the Client to use the Push scheme, the RS MUST respond with an error message using the response code 4.01 (Unauthorized). The error message MUST contain information that allows the Client to locate the AS responsible for the requested resource, and other information needed to make an Authorization Request. For example:

```
4.01 Unauthorized
Content-Format: application/json
["oscoap", "as.example.com/authorize"]
```

The response payload MAY be protected, e.g. signed with the private key of RS, using COSE Mode:PAYL. The response MAY include authentication information of AS, such as the (hash of the) public key of AS. The public key of RS or AS be retrieved in different ways e.g. from a Resource Directory

The initial exchange described in this section may be omitted, and the client can start with the Authorization Request to AS as described in Section 8.2 if the Client has obtained information about the relevant AS in some other way, e.g. using a Resource Directory.

If the RS has access to authorization information about the Client, but it does not apply to the requested resource, the RS MUST answer with an error message using the response code 4.03 (Forbidden). If the RS has authorization information for the Client that applies to the requested resource, but that does not cover the requested action, the RS MUST reply with an error message using the response code 4.05 (Method Not Allowed).

8.2. Authorization Request

Either the Client (Push scheme) or the RS (Pull scheme) can POST an request to the /authorize resource at the AS, specifying the Client's request(s) for authorization using the ARF defined in Section 5.

Upon receiving a POST request to /authorize, the AS MUST perform the following steps:

- Ensure that the request was received over a secure channel (DTLS) or uses object security. If object security was used the AS MUST perform the necessary verifications.

- Ensure that the requesting party is authenticated. This is either the Client (as described in Section 4.2) or the RS (as described in Section 4.1). For DTLS this is supposed to have happened during the handshake, for object security this is accomplished by a signature or MAC over the request, produced by the requesting party.

- Control that the requesting party is authorized to submit this kind of authorization request

If the integrity verification fails, the AS MUST respond with an error message using the response code 4.00 (Bad Request)

If the requesting party is not correctly authenticated, the AS MUST respond with an error message using the response code 4.01 (Unauthorized).

If the requesting party is not authorized to perform this request for authorization information, the AS MUST respond with an error message using the response code 4.03 (Forbidden).

If the evaluation is successful the AS MUST respond with a 2.05 (Content) message code. The response can contain one of the following payloads:

- If any part of the access requested by the Client was not authorized, the payload MUST be empty.
- If all parts of the access requested by the Client were authorized, the payload MUST be either in AIF, CIF or both, depending on the authorization scheme:

1. If the Authorization Scheme is Pull, then the payload MUST be in AIF, as defined in Section 6.
2. If the Authorization Scheme is Client-Pull, then the payload MUST be in CIF as defined in Section 7.
3. If the Authorization Scheme is Push, then the payload MUST be an array containing first CIF, then AIF, as defined in Section 7 and Section 6 respectively.

8.3. Receiving Authorization Information

The RS may receive authorization information in a response to a POST to an Authorization Request as described in the previous section.

As an alternative this section describes how to push this information to the RS. This is done with a POST of the authorization information to the /authorize resource on the RS. This request may come from the Client in the Pull scheme, as defined in Pattern 3 in Section 4.1. The request may also come from the AS directly.

Figure 12 illustrates the successful execution of such a request.

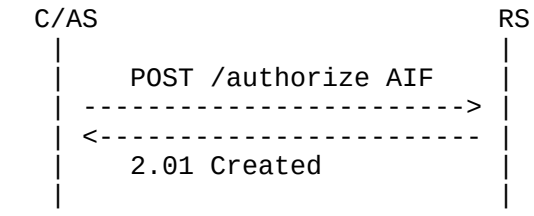


Figure 12: POST AIF to /authorize

A third OPTIONAL method is to include the authorization information in a DTLS handshake as supplemental data, as described in Section 4.4.

Irrespective of how the authorization information is received by the AS, upon receiving it, RS MUST verify that it is valid, by doing the following:

- Check that the authorization information comes from a trusted AS
- Check that the host in the authorization information corresponds to itself
- Check that the authorization information is not expired or revoked, if the object security encapsulation allows this
- Verify the integrity of the authorization information

If the validation succeeds, RS MUST store the authorization information and use it to determine the authorization of future requests.

If the authorization information is invalid, RS MUST respond with an error message using the error code 4.03 (Forbidden).

8.4. Receiving Client Information

The Client receives client information in a response to a POST of an authorization request to an AS as described in Section 8.2. The Client MUST make sure that it is communicating with a trusted AS. If the communication is protected with COSE Mode:COAP, the Client MUST verify the authenticity and integrity of the client information, before using it.

8.5. Resource Request and Response

The final resource request is handled differently based on which authorization scheme is implemented. In the Pull scheme this is an unauthorized request as described in Section 8.1. For resources configured to use the Client-Pull scheme for GET access, the RS MUST unconditionally provide an encrypted and signed representation of the requested resource to any requesting Client.

In the Push scheme the RS MUST perform the following steps:

- If the communication is secured with DTLS, verify that the handshake included client authentication. If the communication is protected with Mode:COAP, verify the object security of the request.
- Verify the present authorization information to see if it has data that matches the Client, the requested resource, and the actions

requested on that resource. Note specifically that the binding between the authorization information and the Client is provided either by comparing with the keys used in the DTLS handshake, or with the keys used for object security of the request.

- o If some matching authorization information contains local conditions, verify that these are fulfilled.

If any of these verifications fail, the RS treat the request as specified in Section 8.1.

If all of these verifications succeed, the RS MUST process the request as required by the underlying application.

If the request was protected with Mode:COAP, the RS MUST follow the response protecting scheme for this mode.

9. Security Considerations

The entire document is about security. Security considerations applicable to authentication and authorization in RESTful environments provided in OAuth 2.0 [RFC6749] apply to this work, as well as the security considerations from [I-D.gerdes-ace-actors].

10. IANA Considerations

This document has no actions for IANA yet.

11. Acknowledmgents

Some of the ideas of this document were originally described in a now expired Internet Draft: draft-selander-core-access-control-01. The authors would also like to thank Carsten Bormann, Stefanie Gerdes and Olaf Bergmann for the inspiration drawn from draft-bormann-core-ace-aif [I-D.bormann-core-ace-aif] and [I-D.gerdes-ace-dcaf-authorize].

This work has further drawn inspriation from [OSCAR] which presents another scheme for securing resource request and response using object security and where access key transport is performed by means of RESTful requests to dedicated resources.

12. References

12.1. Normative References

[I-D.schaad-cose-msg]
Schaad, J., "CBOR Encoded Message Syntax", draft-schaad-cose-msg-00 (work in progress), June 2015.

[I-D.selander-ace-object-security]
Selander, G., Mattsson, J., and L. Seitz, "March 9, 2015", draft-selander-ace-object-security-01 (work in progress), March 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, October 2006.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7250] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, June 2014.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

12.2. Informative References

[I-D.bormann-core-ace-aif]
Bormann, C., "An Authorization Information Format (AIF) for ACE", draft-bormann-core-ace-aif-02 (work in progress), March 2015.

[I-D.gerdes-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-gerdes-ace-actors-05 (work in progress), April 2015.

[I-D.gerdes-ace-dcaf-authorize]
Gerdes, S., Bergmann, O., and C. Bormann, "Delegated CoAP Authentication and Authorization Framework (DCAF)", draft-gerdes-ace-dcaf-authorize-02 (work in progress), March 2015.

[I-D.ietf-ace-usecases]
Seitz, L., Gerdes, S., Selander, G., Mani, M., and S. Kumar, "ACE use cases", draft-ietf-ace-usecases-04 (work in progress), June 2015.

[OSCAR] Vucinic, M., Tourancheau, B., Rousseau, F., Duda, A., Damon, L., and R. Guizzetti, "OSCAR: Object security architecture for the Internet of Things", Ad Hoc Networks Vol. 32, September 2015.

[RFC2904] Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., and D. Spence, "AAA Authorization Framework", RFC 2904, August 2000.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013.

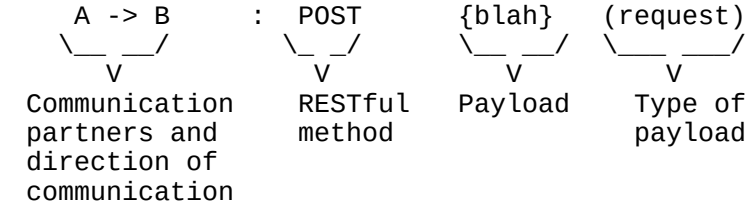
[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, May 2014.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

Appendix A. Examples

The following examples show an overview the different authorization schemes presented in Section 3. The notation is as follows:



Messages protected with COSE Mode:COAP are denoted by adding 'CoseC(Key)', where 'Key' is the key used to protect the message.

Payload protected with COSE Mode:PAYL is denoted by 'CoseP(Key){payload}', where 'Key' is the key used to protect the payload.

Note that we use POST to request authorization information instead of GET, since with the latter, the request would need to be transported in CoAP options (e.g. Uri-Query), and it could potentially become larger than the maximum CoAP packet size. Since CoAP options can not be fragmented this would be extremely problematic.

Example A (using the Push scheme)

1. C -> RS : GET /sensors/tempC {} (request)

2. RS -> C : 4.01 Unauthorized {"oscoap", "as.example.com/authorize"} (response)

3. C -> AS : POST /authorize CoseC(Key_C-AS) { [{"subjectKeyId":"Key_C-AS"}, "rs.example.com", [['sensors/tempC', 1(= GET)]]] } (ARF)

4. AS -> C : 2.05 CoseC(Key_C-AS) { ["push", "oscoap", "/authorize", "psk", "someKeyId", "Z4LXBNbOeeOJOYglBLb4pg"], CoseP(Key_AS-RS){ [{"SubjectKeyId":"someKeyId"}, "rs.example.com", [["sensors/tempC", 1(= GET)]]] } } (CIF, AIF)

5. C -> RS : POST /authorize CoseC(someKeyId) { CoseP(Key_AS-RS){[{"SubjectKeyId":"someKeyId"}, "rs.example.com", [["sensors/tempC", 1(= GET)]]]} } (AIF)

6. C -> RS : GET /sensors/tempC CoseC(someKeyId) {} (request)

7. RS -> C : 2.05 CoseC(someKeyId) {37.5} (response)

Example B (using the Pull scheme)

1. C -> RS : POST /actuators/doorLock CoseC(someClientKey) {open} (request)

2. RS -(DTLS)-> AS : POST /authorize { [{"subjectKeyId":"someClientKey"}, "rs.example.com", [['actuators/doorLock', 2(= POST)]]] } (ARF)

3. AS -(DTLS)-> RS : 2.05 { [{"subjectKeyId":"someClientKey"}, "rs.example.com", [['actuators/doorLock', 2(= POST)]]] } (AIF)

4. RS -> C : 2.04 (response)

Example C (using the Client-Pull scheme)

1. C -> RS : GET /sensors/tempC {} (request)

2. RS -> C : 2.05 { CoseP(someKey){39.2} } (response)

3. C -> AS : POST /authorize CoseC(Key_C-AS) { [{"subjectKeyId ":"Key_C-AS"}, "rs.example.com", [[’sensors/tempC’, 1(= GET)]]] } (ARF)

4. AS -> C : 2.05 CoseC(Key_C-AS) { ["cpull", "RS_PublicKey", "someKey] } (CIF)

The authorizon schemes defining these flows are introduced in Section 3, and detailed in Section 8. The message structures for ARF, AIF and CIF are introduced in Section 5, Section 6, and Section 7.

Authors’ Addresses

Ludwig Seitz
SICS
Scheelevaegen 17
Lund 223 70
SWEDEN

Email: ludwig@sics.se

Goeran Selander
Ericsson
Faroegatan 6
Kista 164 80
SWEDEN

Email: goran.selander@ericsson.com

Malisa Vucinic
STMicroelectronics
850 Rue Jean Monnet
Crolles 38920
FRANCE

Email: malisa.vucinic@st.com

ace
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

A. Somaraju
Tridonic GmbH & Co KG
S. Kumar
Philips Research
H. Tschofenig
ARM Ltd.
W. Werner
Werner Management Services e.U.

October 31, 2016

Security for Low-Latency Group Communication
draft-somaraju-ace-multicast-02.txt

Abstract

Some Internet of Things application domains require secure group communication. This draft describes procedures for authorization, key management, and securing group messages. We specify the usage of object security at the application layer for group communication and assume that CoAP is used as the application layer protocol. The architecture allows the usage of symmetric and asymmetric keys to secure the group messages. The asymmetric key solution provides the ability to uniquely authenticate the source of all group messages and this is the recommended architecture for most applications. However, some applications have strict requirements on latency for group communication (e.g. in non-emergency lighting applications) and it may not always be feasible to use the secure source authenticated architecture. In such applications we recommend the use of dynamically generated symmetric group keys to secure group communications.

Table of Contents

1. Introduction

There are low latency group communication use cases that require securing communication between a sender, or a group of senders, and a group of receivers. In the lighting use case, a set of lighting nodes (e.g., luminaires, wall-switches, sensors) are grouped together into a single "Application Group" and the following three requirements need to be addressed:

1. Only authorized members of the application group must be able to read and process messages.

2. Receivers of group messages must be able to verify the integrity of received messages as being generated within the group.

3. Message communication and processing must happen with a low latency and in synchronous manner.

This document discusses a group communication security solution that satisfies these three requirements. As discussed in Section 4, we recommend the usage of an asymmetric key solution that allows unique source authentication of all group messages. However, in situations where the low latency requirements can not be met (e.g. in non-emergency lighting applications), the alternative architecture discussed in Section 3 based on symmetric keys is recommended.

2. Terminology

This document uses the following terms from [I-D.ietf-ace-actors]: Authorization Server, Resource Owner, Client, Resource Server. The terms 'sender' and 'receiver' refer to the application layer messaging used for lighting control; other communication interactions with the supporting infrastructure uses unicast messaging.

When nodes are combined into groups there are different layers of those groups with unique characteristics. For clarity we introduce terminology for three different groups:

Application Group:

An application group consists of the set of all nodes that have been configured to respond to a single application layer request. For example, a wall mounted switch and a set of luminaires in a single room might belong to a single group and the switch may be used to turn on/off all the luminaires in the group simultaneously

with a single button press. In the remainder of this document we will use GId to identify an application group.

Multicast Group:

A multicast group consists of the set of all nodes that subscribe to the same multicast IP address.

Security Group:

A security group consists of the set of all nodes that have been provisioned with the same keying material. All the nodes within a security group share a security association or a sequence of security associations wherein a single association specifies the keying material, algorithm-specific information, lifetime and a key ID.

Source-authenticated Security Group:

A source-authenticated security group consists of the set of receiver nodes that have been provisioned with the public verification keying material of all the sender nodes and the set of sender nodes that are provisioned with their unique private signing keying material. All the nodes within a source-authenticated security group share a security association or a sequence of security associations wherein a single association specifies the the public or private keying material, algorithm-specific information, lifetime and a key ID.

Typically, the four groups might not coincide due to the memory constraints on the devices and also security considerations. For instance, in a small room with windows, we may have three application groups: "room group", "luminaires close to the window group" and "luminaires far from the window group". However, we may choose to use only one multicast group for all devices in the room and one security group for all the devices in the room. Note that every application group belongs to a unique security group. However, the converse is not always true. This implies that the application group ID maybe used to determine the associated security group but not vice versa.

The fact that security groups may not coincide with application groups implies that

(1) an application must be able to specify which resources on a resource server are accessible by a client that has access to the group key, and

(2) a method is required to associate the group key to the application group(s) for which the group key may be used.

In this document we provide fields that may be used to specify the "scope of the key" and "application groups for which the key may be used". A commissioner has a lot of flexibility to assign nodes to multicast groups and to security groups while the application groups will be determined by the semantics of the application itself. The exact partitioning of the nodes into security and multicast groups is therefore deployment specific.

3. Architecture - Group Authentication

Each node in a lighting application group might be a sender, a receiver or both sender and receiver (even though in Figure 1, we show nodes that are only senders or only receivers for clarity). The low latency requirement implies that most of the communication between senders and receivers of application layer messages is done using multicast IP. On some occasions, a sender in a group will be required to send unicast messages to unique receivers within the same group and these unicast messages also need communication security.

Two logical entities are introduced and they have the following function:

Key Distribution Center (KDC): This logical entity is responsible for generating symmetric keys and distributing them to the nodes authorized to receive them. The KDC ensures that nodes belonging to the same security group receive the same key and that the keys are renewed based on certain events, such as key expiry or change in group membership.

Authorization Server (AS): This logical entity stores authorization information about devices, meta-data about them, and their roles in the network. For example, a luminaire is associated with different groups, and may have meta-data about its location in a building.

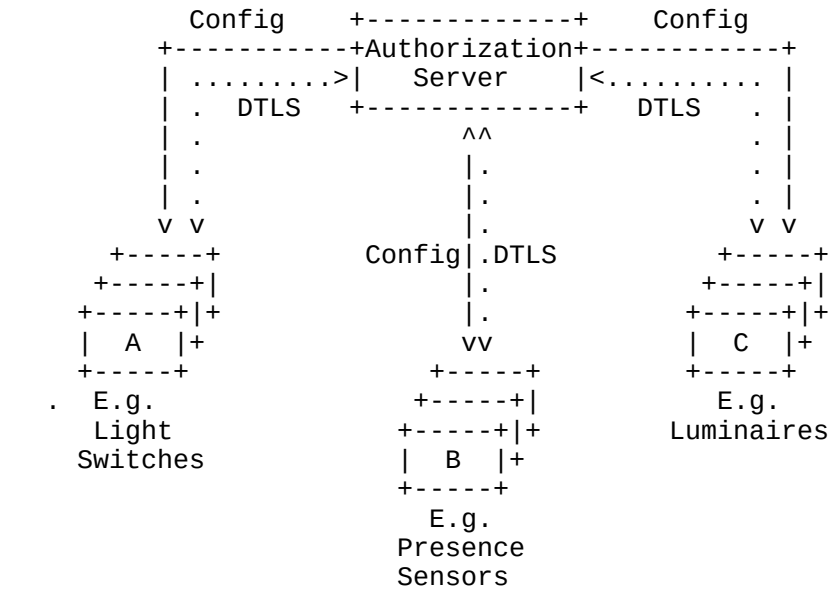
Note that we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 1 and Figure 2 provide an architectural overview. The dotted lines illustrate the use of unicast DTLS messages for securing the message exchange between all involved parties. The secured group messages between senders and receivers are indicated using lines with

star/asterisk characters. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) which are to be defined.

Figure 1 illustrates the information flow between an authorization server and the nodes participating in the lighting network, which includes all nodes that exchange lighting application messages. This step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control lights, may require an authorization server that dynamically executes access control decisions.

Figure 1 shows the commissioning phase where the nodes obtain configuration information, which includes the AT-KDC. The AT-KDC is an access token and includes authorization claims for consumption by the key distribution center. We use the access token terminology from [RFC6749]. The AT-KDC in this architecture may be a bearer token or a proof-of-possession (PoP) token. The bearer token concept is described in [RFC6750] and the PoP token concept is explained in [I-D.ietf-oauth-pop-architecture]. The AT-KDC is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-KDC is protected against modifications using a digital signature or a message authentication code. It is verified in Figure 2 by the KDC.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-KDC) and other meta-data.

Figure 1: Architecture: Commissioning Phase.

In the simplified message exchange shown in Figure 2 a sender requests a security group key and the access token for use with the receivers (called AT-R). The request contains information about the resource it wants to access, such as the application group and other resource-specific information, if applicable, and the previously obtained AT-KDC access token. Once the sender has successfully obtained the requested information it starts communicating with receivers in that group using group messages. The symmetric key obtained from the KDC is used to secure the groups messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary group key to verify the incoming messages and to determine what resource the requestor is authorized to access. Both pieces of information can be found in the AT-R access token.

Group messages need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using

a keyed message digest in combination with the group key. The use of symmetric keys is envisioned in this specification due to latency requirements. For unicast messaging between the group members and the AS or KDC, we assume the use of DTLS for transport security. However, the use of TLS, and application layer security is possible but is outside the scope of this document.

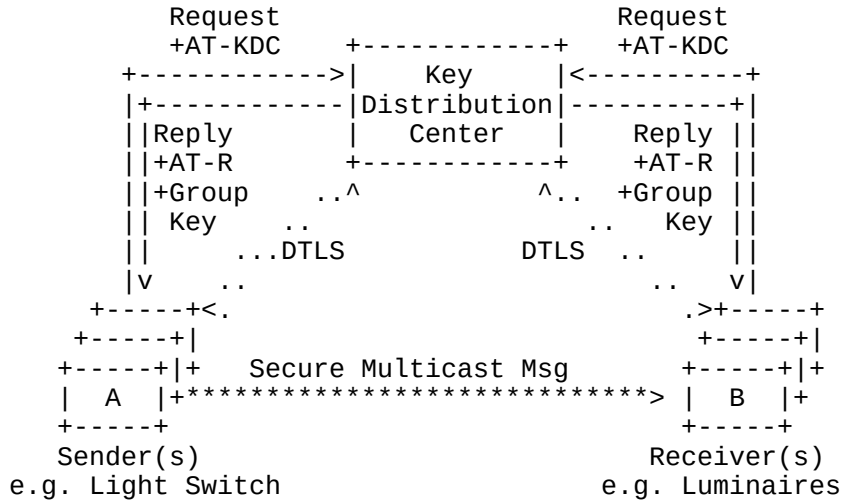


Figure 2: Architecture: Group Key Distribution Phase.

3.1. Assumptions

1. The AT-KDC is a manifestation of the authorization granted to a specific client (or user running a client). The AT-KDC is longer-lived and can be used to request multiple AT-Rs.

2. Each AT-R is valid for use with one or multiple application groups.

3. The AS and the KDC logical roles may reside in different physical entities.

4. The AT-KDC as well as the AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.

5. The AT-KDC token is opaque to the client. Data that is meant for processing by the client has to be conveyed to the client

separately. The AT-R token on the other hand is meant for consumption by the client.

6. The client requests AT-Rs for different application groups by including additional information in the request to the KDC for what application groups the AT-R(s) have to be requested. The KDC may return multiple AT-Rs in a single response (for performance reasons).

7. The AT-KDC and the AT-R are encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

3.2. AT-KDC Access Tokens

The AT-KDC contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Recipient/Audience: Indication to whom the AT-KDC was issued to. In this case, it is the KDC.

5. Client ID: Information about the client that was authenticated by the authorization server.

6. Issued at: Indicates date and time when the AT-KDC was created by the authorization server.

3.3. AT-R Access Tokens

Clients send the AT-KDC to the KDC in order to receive an AT-R.

The KDC MUST maintain a table consisting of scope values, which includes the application group id. These entries point to a sequence of security associations. A security association specifies the key material, algorithm-specific information, lifetime and a key ID and the key ID may be used to identify this security association.

The AS/KDC must guarantee the uniqueness of the client ids for its nodes. This may be accomplished by the AS/KDC assigning values to the nodes or by using information that is already unique per device (such as an EUI-64).

The KDC furthermore needs to be configured with information about the authorization servers it trusts. This may include a provisioned trust anchor store, or shared credentials (similar to a white list).

The KDC MUST generate new group keys after the validity period of the current group key expires.

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Security Group Key: Key to use for the group communication.

5. Algorithm: Used for secure group communication.

6. KID: Sequentially increasing ID of the key for the security group (the devices may store an older key to help with key rolling.)

7. Issued at: Indicates date and time when the AT-R was created by the KDC.

3.4. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

    * Client ID (unencrypted, integrity protected): Every sender managed by a key distribution center MUST have a unique client ID.

   * Sequence Number (unencrypted, integrity protected): Used for replay protection.

   * An implicit IV that is either derived from the keys at the end-points or fixed to a certain value by standard (not sent in the message)

2. MAC (not integrity protected): For integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.

2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.

3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.

4. Key ID (unencrypted, integrity protected): Is used to select the current security association from the sequence of security associations identified by the application group id.

5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)

6. CoAP Payload (encrypted, integrity protected).

3.5. Receiver Algorithm

All receiving devices MUST maintain a table consisting of mappings of application group id, to a sequence of security associations.

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association.

The key id is used for situations where the group key is updated by the KDC (for example in situations where a device in a group is lost or stolen).

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- Application Group ID lookup does not return any security association.
- Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- Integrity check fails.
- Decryption fails.
- Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

3.6. Sender Algorithm

Figure 3 describes the algorithm for obtaining the necessary credentials to transmit a secure group message. When the sender wants to send a message to the application group, it checks if it has the respective group key. If no group key is available then it determines whether it has an access token for use with the KDC (i.e., AT-KDC). If no AT-KDC is found in the cache then it contacts the authorization server to obtain that AT-KDC. Note that this assumes

that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported. In the other case where the AT-KDC is already available the sender contacts the KDC to obtain a group key. If a group key is already available then the sender can transmit a secured message to the group immediately.

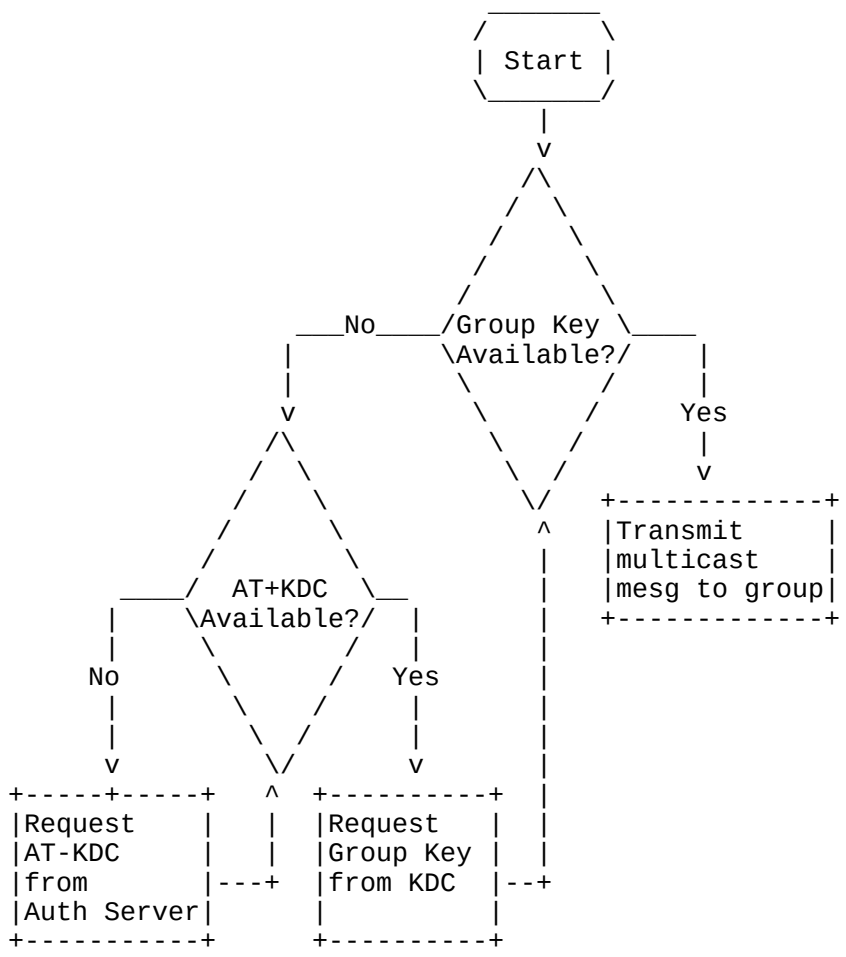


Figure 3: Steps to Transmit Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a group key; the sender is likely to be

pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

Group keys have a lifetime, which is configuration-dependent, but mechanisms need to be provided to update the group keys either via the sender asking for a group key renewal or via the KDC pushing new keys to senders and receivers. The lifetime can be based on time or on the number of transmitted messages.

4. Architecture - source authentication

This section discusses the usage of asymmetric keys to achieve source authentication of group messages and is the recommend architecture for securing group messages. However, this solution may not meet the low latency requirement without adequate hardware support but still most of the group communication between senders and receivers of application layer messages is done using multicast IP.

Unlike the previous architecture, the current architecture requires only the Authorization Server (AS) logical entity as defined in the previous section.

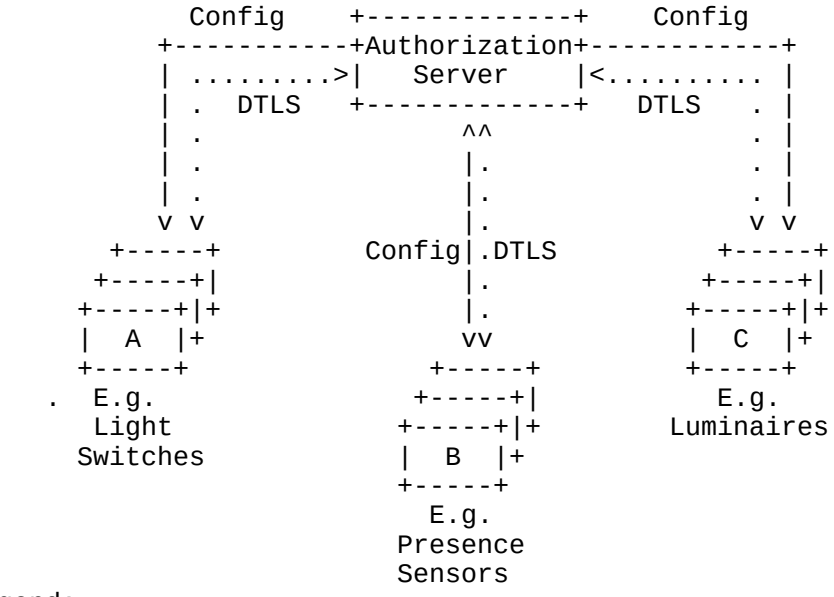
As in the previous case we assume that nodes are pre-configured with device credentials (e.g., a certificate and the corresponding private key) during manufacturing or during an initial provisioning phase. These device credentials are used in the interaction with the authorization server.

Figure 4 and Figure 5 provide an architectural overview for the source authenticated case. The main differences from the previous case is that the AS provides directly the AT-R tokens. Further no KDC is required in this case since the senders and receivers can use their public-private key pair credentials to secure messages. The AS may provide authorization based on the pre-existing device credentials or issue new credentials to the devices. The security of the group messages is accomplished at the application level using small modification to OSCOAP - Object Security of CoAP (see [I-D.selander-ace-object-security]) but based on public key signatures which are to be defined.

Figure 4 illustrates the information flow between an authorization server and the nodes participating in the source-authenticated group network. Like the previous case, this step is typically executed during the commissioning phase for nodes that are fixed-mounted in buildings. The authorization server, as a logical function, may in smaller deployments be included in a device carried by the commissioner and only be present during the commissioning phase. Other use cases, such as employees using their smartphones to control

lights, may require an authorization server that dynamically executes access control decisions.

Figure 4 shows the commissioning phase where the nodes obtain configuration information, which includes directly the AT-R. The AT-R is an access token and includes authorization claims for consumption by the receivers. The AT-R may be a bearer token or a proof-of-possession (PoP) token. The AT-R is created by the authorization server after authenticating the requesting node and contains authorization-relevant information. The AT-R is protected against modifications using a digital signature. It is verified in Figure 5 by the receivers.



Legend:

Config (Configuration Data): Includes configuration parameters, authorization information encapsulated inside the access token (AT-R) and other meta-data.

Figure 4: Architecture - Source-authenticated: Commissioning Phase.

In the simplified message exchange shown in Figure 5 a sender starts communicating with receivers in that source-authenticated group using public-key signed group messages. The AT-R may be attached to the initial request.

Receivers need to perform two steps, namely to obtain the necessary public verification key of the senders (or a root verification key if they are certified by the same authority) to verify the incoming messages and the public verification key of the AS to determine what resource the requestor is authorized to access. Both pieces of information can either be found in the AT-R access token or separately configured during the commissioning phase.

Source-authenticated Group messages also need to be protected such that replay and modification can be detected. The integrity of the message is accomplished using a public-key signature. This may not achieve the latency requirements and used where source-authentication is more important. For unicast messaging between the group members and the AS , we assume the use of DTLS for transport security.

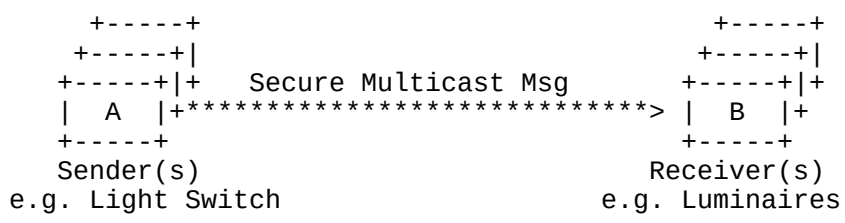


Figure 5: Architecture - Source-authenticated: Group communication.

4.1. Assumptions

1. The AT-R is a manifestation of the authorization granted to a specific client (or user running a client). The AT-R is longer-lived and can be used directly for source-authenticated group communication until it is revoked or expired.

2. Each AT-R is valid for use with one or multiple application groups.

3. The AT-R may be self-contained tokens or references. References are more efficient from a bandwidth point of view but require an additional lookup.

4. The AT-R token is not opaque to the client and is meant for consumption by the client.

5. The client requests AT-Rs for different application groups by including additional information in the request to the AS for what application groups the AT-R(s) have to be requested. The AS

may return multiple AT-Rs in a single response (for performance reasons).

6. The AT-R is encoded as CBOR Web Tokens [I-D.wahlstroem-ace-cbor-web-token] and protected using COSE [I-D.ietf-cose-msg].

4.2. AT-R Access Tokens

The AT-R contains

1. Issuer: Entity creating the access token. This information needs to be cryptographically bound to the digital signature/keyed message digest protecting the content of the token, as provided by the CBOR Web Token (CWT).

2. Expiry date: Information can be omitted if tokens do not expire (for example, in a small enterprise environment).

3. Scope: Permissions of the entity holding the token. This includes information about the resources that may be accessed with the token (e.g., access level) and application layer group IDs for the groups for which the tokens may be used.

4. Recipient/Audience: Indication to whom the AT-R was issued to. In this case, it is the receivers.

5. Client ID: Information about the client that was authenticated by the authorization server.

6. Client public key: The public key to use for signing the source-authenticated group communication. These public key may be optionally certified using the AS key or a domain root key. This reduces the need for additional per-device public key storage on the receivers.

7. Algorithm: Used for source-authenticated secure group communication.

8. Issued at: Indicates date and time when the AT-R was created by the authorization server.

4.3. Multicast Message Content

The following information is needed for the cryptographic algorithm, which is assumed to be in the COSE header:

1. Nonce value consisting of

   * Client ID (unencrypted, integrity protected): Every sender managed by the AS MUST have a unique client ID.

   * Sequence Number (unencrypted, integrity protected): Used for replay protection.

2. Signature (not integrity protected): For source-authenticated integrity protection.

The following information is additionally required to process the secure message:

1. Destination IP address and port (not encrypted, integrity protected): Integrity protection of the IP address and port ensures that the message content cannot be replayed with a different destination address or on a different port.

2. CoAP Path (encrypted, integrity protected): Uniquely identifies the target resource of a CoAP request.

3. Application Group id in CoAP header (unencrypted, integrity protected): Is used to identify a sequence of security associations to use to decrypt the message. The CoAP header option is TBD.

4. Key ID (unencrypted, integrity protected): Is used to select the correct security association containing the verification key from the sequence of security associations identified by the application group id.

5. CoAP Header Options other than application group id (encrypted - if desired, integrity protected)

6. CoAP Payload (encrypted, integrity protected).

4.4. Receiver Algorithm

When a node receives an incoming multicast message it looks up the application group id and the key id (which are both found in the CoAP header) to determine the correct security association to use to verify the message.

The key id is used for situations where the client may have different keys for different applications.

To check for replay attacks the receiver has to consult the state stored with the security association to obtain the current sequence number and to compare it against the sequence number found in the

request payload for that sender based on the Sender ID. The receiver needs to store the latest correctly verified nonce values to detect replay attacks

The receiver MUST silently discard an incoming message in the following cases:

- Application Group ID lookup does not return any security association.
- Key ID lookup among the previously retrieved sequence of security associations does not identify a unique security association.
- Integrity check fails.
- Replay protection check failed. The (client ID || sequence number), which are both part of the nonce, have already been received in an earlier message.

Once the cryptographic processing of the message is completed, the receiver must check whether the sender is authorized to access the protected resource, indicated by the CoAP request URI at the right level. For this purpose the receiver consults the locally stored authorization database that was populated with the information obtained via the AT-R token and the static authorization levels described in Appendix A.

Once all verification steps have been successful the receiver executes the CoAP request and returns an appropriate response. Since the response message will also be secured the message protection processing described in Section 3.6 must be executed. Additionally, the nonce value corresponding to the security association MUST be updated to the nonce value in the message.

4.5. Sender Algorithm

Figure 6 describes the algorithm for obtaining the necessary credentials to transmit a source-authenticated secure group message. When the sender wants to send a message to the application group, it checks if it has the respective signing key that matches the KID in the AT-R. If no signing key is available then it contacts the authorization server to obtain the AT-R and corresponding signing keys. Note that this assumes that the authorization server is online, which is only true in scenarios where granting authorization dynamically is supported.

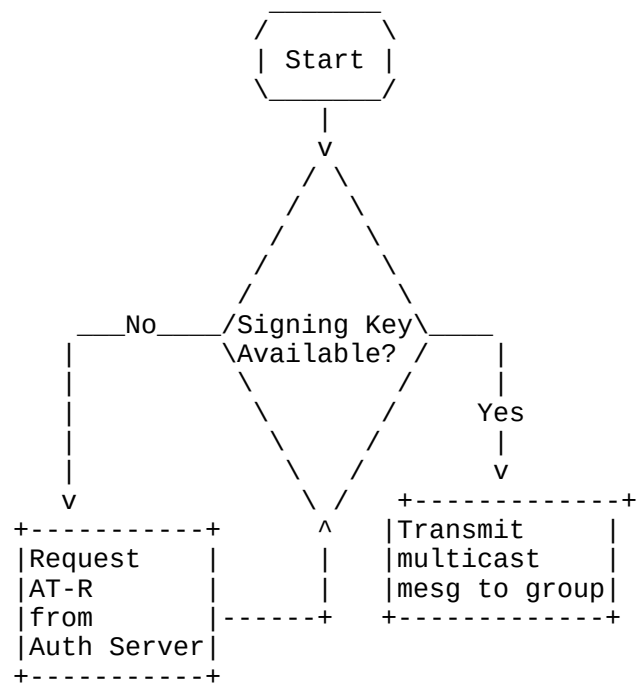


Figure 6: Steps to Transmit Source-authenticated Multicast Message (w/o Failure Cases).

Note that the sender does not have to wait until it has to transmit a message in order to request a AT-R; the sender is likely to be pre-configured with information about which application group it belongs to and can therefore pre-fetch the required information.

5. Security Considerations

5.1. Applicability statement

This document describes two architectures based on symmetric group keys in Section 3 and asymmetric keys in Section 4.

The symmetric key solution is based on a group key that is shared between all group members including senders and receivers. As all members of the group posses the same key, it is only possible to authenticate group membership for the source of a message. In particular, it is not possible to authenticate the unique source of a message and consequently it is not possible to authorize a single

node to control a group. Moreover, because the group key is shared across multiple nodes, it may be easier for an attacker to determine the group key by attacking any member of the group (note that this group key is dynamically generated and is usually stored in volatile memory which offers some addition protection). Subsequent to such an attack, it is also difficult to determine which of the group members was compromised and this makes it difficult to return the system to normal operation after an attack.

The asymmetric key solution distinguishes between a sender in the group and the receivers. In particular, the sender is in possession of a private key and the receivers are in possession of the corresponding public key. This allows the unique source of any group message to be authenticated. Moreover, an attacker cannot compromise the system by breaking into any of the receiving nodes. However, for constrained devices, the asymmetric key solution comes at a processing cost with cryptographic computations taking too long.

Therefore, it is recommended that whenever possible, the architecture with source authentication SHOULD be used to secure all multicast communication. However, in less sensitive applications (e.g. controlling luminaires in non-emergency applications), the architecture without source authentication MAY be used. When using the symmetric key solution two mitigating factors could improve system security. It is possible to achieve source authentication of messages at lower layers by requiring unique MAC layer keys for all devices within the network. The symmetric group keys are dynamically generated and therefore SHOULD be stored in volatile memory.

5.2. Token Verification

Due to the low latency requirements, token verification needs to be done locally and cannot be outsourced to other parties. For this reason a self-contained token must be used and the receivers are required to follow the steps outlined in Section 7.2 of RFC 7519 [RFC7519]. This includes the verification of the message authentication code protecting the contents of the token and the encryption envelope protecting the contained symmetric group key.

5.3. Token Revocation

Tokens have a specific lifetime. Setting the lifetime is a policy decision that involves making a trade-off decision. Allowing a longer lifetime increases the need to introduce a mechanism for token revocation (e.g., a real-time signal from the KDC/Authorization Server to the receivers to blacklist tokens) but lowers the communication overhead during normal operation since new tokens need to be obtained only from time to time. Real-time communication with

the receivers to revoke tokens may not be possible in all cases either, particularly when off-line operation is demanded or in small networks where the AS or even the KDC is only present during commissioning time.

We therefore recommend to issue short-lived tokens for dynamic scenarios like users accessing the lighting infrastructure of buildings using smartphones, tablets and alike to avoid potential security problems when tokens are leaked or where authorization rights are revoked. For senders that are statically mounted (like traditional light switches) we recommend a longer lifetime since re-configurations and token leakage is less likely to happen frequently.

To limit the authorization rights, tokens should contain an audience restriction, scoping their use to the intended receivers and to their access level.

5.4. Time

Senders and receivers are not assumed to be equipped with real-time clocks but these devices are still assumed to interact with a time server. The lack of accurate clocks is likely to lead to clock drifts and limited ability to check for replays. For those cases where no time server is available, such as in small network installations, token verification cannot check for expired tokens and hence it might be necessary to fall-back to tokens that do not expire.

6. Operational Considerations

6.1. Persistence of State Information

Devices in the lighting system can often be powered down intentionally or unintentionally. Therefore the devices may need to store the authorization tokens and cryptographic keys (along with replay context) in persistent storage like flash. This is especially required if the authorization server is no more online because it was removed after the commissioning phase. However the decision on the data to be persistently stored is a trade-off between how soon the devices can be back online to normal operational mode and the memory wear caused due to limited program-erase cycles of flash over the 15-20 years life-time of the device.

The different data that may need to be stored are access tokens AT-KDC, AT-R and last seen replay counter.

6.2. Provisioning in Small Networks

In small networks the authorization server and the KDC may be available only temporarily during the commissioning process and are not available afterwards.

6.3. Client IDs

A single device should not be managed by multiple KDCs. However, a group of devices in a domain (such as a lighting installation within an enterprise) should either be managed by a single KDC or, if there are multiple KDCs serving the devices in a given domain, these KDCs MUST exchange information so that the assigned client id and application group id values are unique within the devices in that domain. We assume that only devices within a given domain communicate with each other using group messages.

6.4. Application Groups vs. Security Groups

Multiple application groups may use the same key for performance reasons, reducing the number of keys needed to be stored - leading to less RAM needed by each node. This is only a reasonable option if the attack surface is not increased. For example, a room A is configured to use three application groups to address a subset of the device. In addition to configuring all nodes in room A with these three application groups the nodes are configured with a special group that allows them to access all devices in room A, referred as the all-nodes-in-room-A group. In this case, having the nodes to use the same key for the all-nodes-in-room group and the three groups does not increase the attack surface since any node can already use the all-nodes-in-room-A group to control other devices in that room. The three application groups in room A are a subset of the larger all-nodes-in-room-A group.

6.5. Lost/Stolen Device

The following procedure MUST be implemented if a device is stolen or keys are lost.

1. The AS tells the KDC to invalidate the AT-KDC.

2. The KDC no longer returns a new group key if the invalidated AT-KDC is presented to it.

3. The KDC generates new keys for all security groups to which the compromised device belongs.

The KDC SHOULD inform all devices in the security group to update their group key. This requires the KDC to maintain a list of all devices that belong to the security group and to be able to contact them reliably.

7. Acknowledgements

The author would like to thank Esko Dijk for his help with this document.

Parts of this document are a byproduct of the OpenAIS project, partially funded by the Horizon 2020 programme of the European Commission. It is provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of fitness for a particular purpose. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the OpenAIS project or the European Commission.

8. IANA Considerations

This document defines one CoAP Header Option Application Group ID that MUST be allocated in the Registry "CoAP Option Numbers" of [RFC6749]. IANA is requested to allocation TBD option number to application group ID in this specification.

9. References

9.1. Normative References

[I-D.ietf-ace-actors]
Gerdes, S., Seitz, L., Selander, G., and C. Bormann, "An architecture for authorization in constrained environments", draft-ietf-ace-actors-04 (work in progress), September 2016.

[I-D.ietf-cose-msg]
Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-23 (work in progress), October 2016.

[I-D.wahlstroem-ace-cbor-web-token]
Wahlstroem, E., Jones, M., and H. Tschofenig, "CBOR Web Token (CWT)", draft-wahlstroem-ace-cbor-web-token-00 (work in progress), December 2015.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

9.2. Informative References

[I-D.ietf-oauth-pop-architecture] Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-08 (work in progress), July 2016.

[I-D.selander-ace-object-security] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-selander-ace-object-security-06 (work in progress), October 2016.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

Appendix A. Access Levels

A characteristic of the lighting domain is that access control decisions are also impacted by the type of operation being performed and those categories are listed below. The following access levels are pre-defined.

Level 0: Service detection only

This is a service that is used with broadcast service detection methods. No operational data is accessible at this level.

Level 1: Reporting only

This level allows access to sensor and other (relatively uncritical) operational data and the device error status. The operation of the system cannot be influenced using this level.

Level 2: Standard use

This level allows access to all operational features, including access to operational parameters. This is the highest level of access that can be obtained using (secure) multicast.

Level 3: Commissioning use / Parametrization Services

This level gives access to certain parameters that change the day-to-day operation of the system, but does not allow structural changes.

Level 4: Commissioning use / Localization and Addressing Services

(including Factory Reset) This level allows access to all services and parameters including structural settings.

Level 5: Software Update and related Services

This level allows the change and upgrade of the software of the devices.

Note: The use of group security is disallowed for level higher than Level 2 and unicast communication is used instead.

Authors' Addresses

Abhinav Somaraju
Tridonic GmbH & Co KG
Farbergasse 15
Dornbirn 6850
Austria

Email: abhinav.somaraju@tridonic.com

Sandeep S. Kumar
Philips Research
High Tech Campus 34
Eindhoven 5656 AE
Netherland

Email: ietf.author@sandeep-kumar.org

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Walter Werner
Werner Management Services e.U.
Josef-Anton-Herrburgerstr. 10
Dornbirn 6850
Austria

Email: werner@werner-ms.at